POLITECNICO DI TORINO

Collegio di Ingegneria Chimica e dei Materiali

Corso di Laurea Magistrale in Ingegneria Chimica e dei Processi Sostenibili

Tesi di Laurea Magistrale

Sviluppo di un metodo multifluido per la simulazione di sistemi gas-liquido in OpenFOAM

Relatori

Prof. Antonio Buffo
Prof. Marco Vanni

Candidato

Sabrina Ortu

LUGLIO 2018

Ai miei genitori

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Simulazione CFD di sistemi multifase

I sistemi multifase si incontrano in molti processi nelle industrie chimiche, petrolifere, farmaceutiche, biochimiche, alimentari ed energetiche. Caratteristica peculiare di questi sistemi è la presenza di una superficie di discontinuità, nota come interfaccia, che può assumere configurazioni molto complesse e subire fenomeni di instabilità e rottura, a causa dell'interazione fluidodinamica fra le fasi.
I flussi multifase sono quindi dei sistemi caratterizzati da due o più fasi immiscibili. Esempi di questi sono i sistemi: gas-liquido, gas-solidi, liquido-solido, liquido-liquido e trifase. Inoltre questi flussi possono assumere diverse configurazioni: ad esempio i flussi gas-liquido possono essere sottoforma di bolle di gas disperse in liquido, oppure gocce di liquido disperse in gas, oppure ancora si possono avere flussi separati in cui si ha un flusso continuo di liquido e di gas. I primi due esempi sono noti come flussi dispersi, poiché una fase è dispersa e l'altra è continua. I flussi gas-solido più comuni sono costituiti da una fase gassosa con particelle solide sospese, questi sistemi si utilizzano tipicamente nei letti fluidizzati o nel trasporto pneumatico. I flussi liquido-solido comunemente utilizzati sono costituiti da una fase solida dispersa in una fase liquida. I flussi liquido-liquido sono formati da due fasi liquide immiscibili, come le emulsioni in cui un liquido sotto forma di minutissime goccioline è disperso in un altro liquido. I flussi trifase solido-liquido-gas, invece, sono molto complessi, poiché coinvolgono tre fasi differenti, e si incontrano ad esempio in applicazioni caratterizzate dalla presenza di bolle nel trasporto dei fanghi [19].
Uno strumento molto utile per lo studio dei sistemi multifase è la fluidodinamica computazionale (CFD), impiegata nel campo ingegneristico in fase di progettazione e scale-up. La simulazione numerica di flussi multifase è stata oggetto di grande interesse negli ultimi anni, sia per il ruolo che essi rivestono in molte applicazioni ingegneristiche, sia per i numerosi fenomeni naturali in cui coesistono due o più fasi, come ad esempio quelli che si incontrano negli studi meteorologici.
L'utilizzo della CFD riduce l'onerosità di campagne sperimentali, consentendo di analizzare l'influenza di differenti condizioni operative, di variazioni geometriche o di valutare l'effetto delle singole variabili sul processo globale.
Di contro, l'impiego della modellazione CFD per sistemi multifase risulta molto limitato rispetto ai sistemi monofase o ai sistemi multifase diluiti. Infatti un flusso monofase laminare può essere simulato in modo molto accurato e per la maggior parte dei flussi monofase turbolenti le simulazioni sono affidabili. Tuttavia, la maggior parte dei sistemi trattati in campo ingegneristico sono multifase e le simulazioni di questo tipo di sistemi

al momento presentano ancora oggi difficoltà di diversa natura.
Fin dagli anni '70 numerosi gruppi di ricerca hanno focalizzato la loro attività sulla messa a punto di metodi numerici per lo studio della dinamica di questi sistemi, dando così vita ad una vasta letteratura scientifica.
Esistono essenzialmente tre approcci: il metodo euleriano-lagrangiano che considera le bolle come entità individuali tracciate usando equazioni che risolvono la loro traiettoria [24], il metodo euleriano-euleriano che considera sia la fase continua che quella dispersa come fluidi continui e interpenetranti, e i metodi per il tracciamento dell'interfaccia che risolvono il profilo dell'interfaccia su scale delle dimensioni delle particelle. Le equazioni di base sono state formulate per la prima volta nel 1975 da Ishii [11].
I modelli euleriani-euleriani sono stati ampiamente studiati, e sono diventati utili per la modellazione dei flussi in applicazioni industriali, poiché consentono di equilibrare il costo computazionale delle simulazioni con la precisione nella descrizione dei flussi.
Diversi autori utilizzarono questo tipo di approccio, per citarne alcuni: Sokolichin ed Eigenberger [21] nel 1997 analizzarono l'utilizzo dei due differenti approcci per sistemi gas-liquido; Pfleger et al. [18] nel 1999 utilizzarono un approccio euleriano-euleriano per lo studio della fluidodinamica delle colonne a bolle; e ancora Buwa et al. [6] nel 2002 analizzarono la dinamica di un flusso gas-liquido all'interno di una colonna a bolle di sezione rettangolare.
Tuttavia, per la complessità intrinseca dei sistemi multifase da un punto di vista sia fisico che numerico, i codici di fluidodinamica computazionale sono ancora in fase di studio [2]. Secondo van Wachem et al. [1], le ragioni della mancanza di conoscenze fondamentali sui flussi multifase sono dovute in primo luogo alla coesistenza di tipi di flusso diversi, all'interno dei quali si possono instaurare diversi regimi di flusso (flusso anulare, a getto, a bolle, a slug ecc.). In secondo luogo le complesse leggi fisiche e la modellazione matematica dei fenomeni che si verificano in presenza di due o più fasi (dinamica dell'interfaccia, coalescenza, rottura, resistenza, ecc.) sono ancora in gran parte non sviluppati. Inoltre le procedure numeriche per la risoluzione delle equazioni che governano i flussi multifase sono estremamente complesse, e molto spesso richiedono costosi algoritmi di calcolo. Quasi tutti i codici CFD estendono le procedure di risoluzione per sistemi monofase e bifase, con le opportune modifiche, ad applicazioni multifase, che producono soluzioni instabili o poco accurate, e richiedono time-step molto piccoli [1].
In precedenza molti autori hanno studiato diverse procedure e metodologie di calcolo per simulare sistemi multifase a partire da modelli bifase già esistenti all'interno del codice, che sono stati modificati e adattati a seconda dei casi trattati. Per citarne alcuni, Carneiro et al. [8] nel 2008 utilizzarono un solutore del codice CFD a licenza libera OpenFOAM, basato su un approccio euleriano-euleriano, per sistemi bifase come base di partenza per implementare un modello dei momenti per simulare i flussi multifase, allo stesso modo Silva et al. [20] implementarono un modello euleriano multifase in OpenFOAM, estendendo il numero di fasi di un solutore bifase.
Per i motivi sopra citati, in questa tesi si è deciso di analizzare l'abilità di un solutore, implementato nel codice CFD OpenFOAM, nel modellare un flusso multifase, effettuando un confronto con il relativo solutore bifase, verificando, quindi, l'accuratezza del modello multifase implementato.

1.2 Colonne a bolle e la loro simulazione CFD

Data la molteplicità dei sistemi multifase, in questa tesi si è deciso di focalizzare l'attenzione sui sistemi dispersi caratterizzati dalla presenza di una fase continua e una o più

fasi disperse. In particolare si è studiato un sistema gas-liquido in cui il liquido costituisce la fase continua e il gas la fase dispersa, ma la metodologia utilizzata è valida anche per sistemi con proprietà fisiche differenti come per i flussi liquido-liquido, solido-liquido o solido-gas.
In questa tesi, il sistema oggetto di studio è una colonna a bolle. Le colonne a bolle sono oggi ampiamente utilizzate in molti processi industriali, data la loro capacità di raggiungere elevate velocità di trasferimento di materia e calore. La fluidodinamica delle colonne a bolle è dominata dal movimento del pennacchio di bolle e dalle strutture vorticose tridimensionali che si vengono a creare nella fase liquida, che modifica continuamente le dimensioni e le posizioni delle bolle [3].
I regimi di flusso che si vengono a creare nelle colonne a bolle dipendono principalmente dalla velocità del gas e dalla distribuzione delle dimensioni delle bolle, e possono essere classificati in tre tipologie [7]:

1. omogeneo;
2. di transizione;
3. eterogeneo.

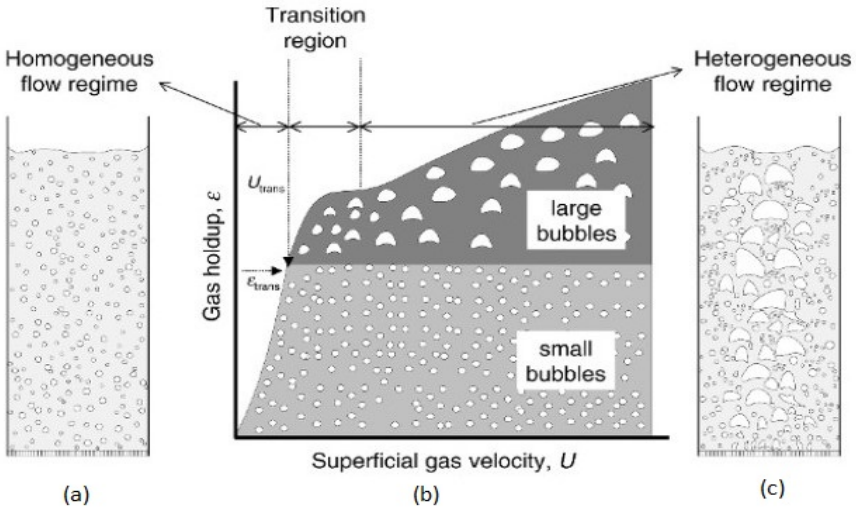

Figura 1.1: Rappresentazione dei regimi di flusso nelle colonne a bolle: (a) regime omogeneo, (b) andamento dell'hold-up di gas in funzione della velocità superficiale del gas per i tre tipi di regime, (c) regime eterogeneo [12].

Il regime omogeneo, mostrato in figura 1.1 (a), si verifica per basse velocità superficiali di gas, ed è caratterizzato da un movimento verticale delle bolle, da una distribuzione dimensionale delle bolle stretta, e da una distribuzione di gas radialmente omogenea.
Per velocità superficiali di gas maggiori si ha il regime di transizione, caratterizzato dalla presenza di un flusso centrale di bolle di dimensioni maggiori che risalgono in superficie creando dei vortici, mentre le bolle più piccole si distribuiscono lateralmente.
Infine, il regime eterogeneo, mostrato in figura 1.1 (c), si ha in presenza di velocità superficiali maggiori rispetto al regime di transizione, e solitamente vengono a crearsi degli agglomerati di bolle che si muovono preferenzialmente nella zona centrale della colonna. In questo caso la distribuzione dimensionale delle bolle è molto larga.
In figura 1.1 (b) è possibile notare come varia l'andamento di hold-up di gas in funzione

della velocità superficiale del gas per i tre tipi di regime. L'hold-up di gas aumenta all'aumentare della velocità superficiale del gas, passando da un regime omogeneo ad uno eterogeneo.
Nei casi in cui le bolle hanno dimensioni diverse, anche le velocità con cui si muovono sono diverse, specialmente se la distribuzione dimensionale è molto ampia, e risulta quindi necessario utilizzare un modello multifluido che non si limiti allo studio di due sole fasi.

1.3 OpenFOAM

OpenFOAM (Open Source Field Operation and Manipulation) è un programma a licenza libera (open source) utilizzato, sia in ambito accademico che industriale, per la risoluzione di un'ampia gamma di problemi fluidodinamici. Più che un software vero e proprio, è da considerarsi come una libreria di codici eseguibili scritti in linguaggio C++, sviluppato e distribuito dalla OpenFOAM Foundation.
Comprende oltre 80 applicazioni solver e 170 applicazioni utilities che svolgono attività di pre e post processing, come meshing, visualizzazione dei dati, estrapolazione di grafici ecc.
I solvers si occupano della risoluzione di specifici problemi della meccanica dei continui (Field Operation), mentre le utilities hanno il compito di manipolare le strutture dati nel pre e post processing (Field Manipulation). I solutori standard coprono una vasta gamma di problemi in fluidodinamica, e includono [10]:

- fluidodinamica computazionale di base;
- flussi incomprimibili;
- flussi comprimibili;
- flussi multifase;
- DNS e LES;
- combustione;
- tracciamento delle particelle;
- trasferimento termico;
- dinamica molecolare;
- simulazione diretta Monte Carlo;
- elettromagnetismo;
- dinamica dei solidi;
- flussi finanziari.

Il programma ha una struttura complessa, che può essere schematizzata come in figura 1.2.
Tuttavia, la documentazione e l'interfaccia utente non sono così esaustive come quelle per i codici commerciali. OpenFOAM è fornito con ambienti pre e post elaborazione, garantendo una gestione facile e coerente dei dati in tutti gli ambienti. In queste fasi è possibile servirsi di software terzi, come ParaView, utilizzato per la visualizzazione e

l'analisi dei dati, che dialoga con OpenFOAM attraverso l'utility paraFoam.
Il fatto di essere un codice CFD a licenza libera, ovvero con codice sorgente disponibile, ne giustifica il largo impiego, sia in ambito accademico che ingegneristico. Una licenza di software libero ne permette l'utilizzo, lo studio e la modifica da parte di chiunque; è infatti possibile avere a disposizione le equazioni e gli algoritmi utilizzati, per essere poi sviluppati e personalizzati. Altri vantaggi di questa tipologia di licenza sono: l'assenza di costi di licenza (specialmente per il calcolo parallelo), la possibilità di customizzazione da parte degli utenti e l'eventuale facilità di collaborazione e di scambio di informazioni in attività di ricerca.

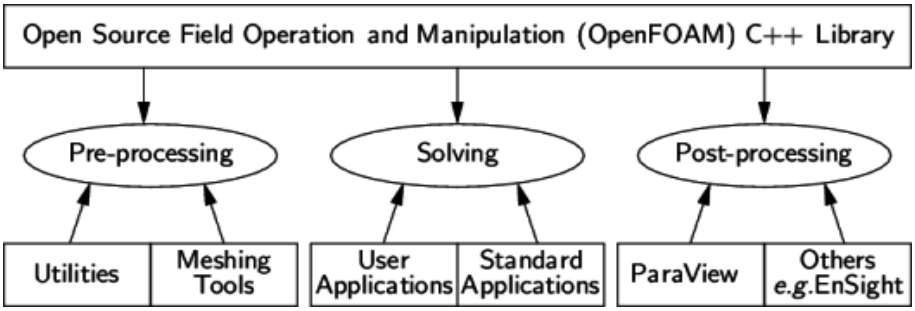

Figura 1.2: Rappresentazione della struttura di OpenFOAM.

1.4 Obiettivo della tesi

L'obiettivo di questo lavoro di tesi è verificare l'utilizzo del solutore `multiphaseEulerFoam` per simulare sistemi multifase con l'approccio multifluido. Per analizzare la capacità del codice, si è operato un confronto con il solutore `twoPhaseEulerFoam`, utilizzato per simulare sistemi con due fasi.
Inizialmente si sono esaminati i due codici nella risoluzione dello stesso sistema, attraverso la realizzazione di test case equivalenti, in modo tale da operare un confronto coerente tra il `twoPhaseEulerFoam` e il `multiphaseEulerFoam`.
Successivamente, si sono analizzate tutte le differenze presenti tra i due codici e tra i modelli teorici e quelli implementati.
Infine si sono descritte tutte le modifiche adottate per poter confrontare i due codici e poter rendere più accurato il modello multifluido implementato.
Il presente lavoro è strutturato come segue.
Nel capitolo 2 è presente un'introduzione teorica sui modelli euleriani, vengono poi descritti il modello Two-fluid per sistemi con due fasi fluide e il modello Multi-fluid per sistemi con più fasi fluide.
Nel capitolo 3 vengono delineate le modalità di implementazione dei modelli euleriani sui codici `twoPhaseEulerFoam` e `multiphaseEulerFoam`. Inoltre è presente una descrizione di tutte le differenze e le modifiche effettuate su entrambi i codici.
Nel capitolo 4 viene descritto il caso in esame con una disamina di tutte le condizioni iniziali e al contorno, le proprietà chimico-fisiche, i parametri di integrazione temporale, e gli schemi numerici adottati nella simulazione.
Infine nel capitolo 5 vengono mostrati i risultati ottenuti inizialmente e con tutte le modifiche attuate successivamente nei codici.

Capitolo 2

Teoria dei modelli euleriani

2.1 Introduzione dei modelli per sistemi multifase

Le metodologie utilizzate per la modellazione di sistemi multifase possono essere molteplici e in generale non esiste un metodo superiore agli altri. Risulta quindi opportuno valutare caso per caso quale sia il metodo più conveniente e più adatto a descrivere la natura fisica del sistema in esame.
In genere è possibile operare una distinzione tra metodi che risolvono il profilo dell'interfaccia su scale delle dimensioni delle particelle, e metodi mediati che non risolvono il profilo dell'interfaccia e vengono applicati all'intera apparecchiatura.
Della prima categoria fanno parte i metodi VOF (Volume of Fluid) nei quali è possibile definire la forma assunta dalle bolle durante il loro movimento, richiedendo una risoluzione della griglia molto elevata e di conseguenza un costo computazionale significativo. In questo caso i due fluidi sono tracciati attraverso una funzione indicatore che in ogni cella fornisce un'informazione sulla presenza di uno o dell'altro fluido, senza considerare la presenza di una miscela.
Altri metodi che tracciano il profilo dell'interfaccia sono i front-tracking (FT) e i level-set (LS). Anche in questi casi i fluidi sono considerati completamente segregati nella cella in cui può aver luogo solo una fase. Nel front-tracking l'interfaccia viene rappresentata tramite dei marker connessi, per formare un fronte che si muove attraverso la griglia; durante questo movimento vengono aggiunti i punti dell'interfaccia. I metodi level-set garantiscono un tracciamento netto dell'interfaccia che viene definita come la curva di livello zero di una funzione continua di pseudo densità. Questi metodi fanno parte della categoria delle Direct Numerical Simulation (DNS), il loro vantaggio è quello di fornire informazioni accurate ma possono essere utilizzate solo per geometrie semplici e di piccole dimensioni.
I metodi mediati possono essere di tipo euleriano-lagrangiano oppure di tipo euleriano-euleriano.
Il primo metodo viene applicato a sistemi come gli spray, in cui le particelle della fase dispersa sono piccole e in quantità non elevate. La fase continua viene trattata tramite un approccio euleriano mentre quella dispersa con quello lagrangiano. La fase dispersa è rappresentata da un numero finito di particelle o gocce, e viene calcolata la traiettoria di ogni singola particella dispersa sulla base di un bilancio di forze che agiscono su di essa. Il numero di particelle il cui moto è tracciato, è solitamente più piccolo del numero attuale di particelle presenti nel fluido.
Tra questi metodi si può menzionare il PSIC (Particle Source in Cell) e il DPM (Discrete

Particle Method). Il primo si utilizza per sistemi gas-liquido o gas-solido con basso valore di frazione volumica della fase liquida o solida, mentre il secondo per alti valori di frazioni volumiche.
Nella formulazione euleriana-euleriana, invece, sia la fase continua che quella dispersa vengono trattate come fluidi interpenetranti e continui, con differenti campi di moto, calcolati tramite le equazioni mediate di Navier-Stokes. La frazione volumica permette di definire il volume occupato da ogni fase su ogni cella, con valori compresi tra 0 e 1.
Tra i metodi mediati euleriani-euleriani, si possono ricordare il modello multifluido e il modello miscela. Il primo utilizza le equazioni in forma mediata per la risoluzione della fisica del sistema, mentre il secondo metodo si occupa di descrivere le proprietà della miscela anziché quelle delle singole fasi.
Riassumendo i metodi DNS permettono di descrivere sistemi gas-liquido con una risoluzione elevata e un accuratezza tale da risolvere ogni aspetto del sistema, compresa l'interfaccia. Di contro utilizzano ingenti risorse di calcolo e non permettono la realizzazione di simulazioni applicabili a sistemi di scala industriale.
I metodi mediati, invece, consentono la risoluzione di sistemi su scale più grandi rispetto alle DNS, con un accettabile costo computazionale.
In questo lavoro di tesi si è scelto di utilizzare il modello multifluido, in grado di descrivere il comportamento di sistemi gas-liquido di scala industriale come colonne a bolle. In questo capitolo verranno descritte le equazioni del modello multifluido per un caso isotermo e senza reazione chimica.

2.2 Modello Multifluido

2.2.1 Modelli mediati

Un sistema multifase è costituito idealmente da un numero definito di regioni, contenente ciascuna una singola fase, separate da interfacce mobili o fisse. Per descrivere un sistema di questo tipo, è possibile partire dalla definizione delle equazioni locali della conservazione della massa, della quantità di moto e dell'energia per ogni fase, rilevando la posizione istantanea dell'interfaccia. Come già accennato, questo procedimento è tipico delle DNS e non è applicabile al caso in esame.
Per procedere, quindi, è possibile utilizzare delle tecniche di mediazione per le equazioni di bilancio locali e istantanee, allo scopo di ottenere delle equazioni macroscopiche in grado di descrivere le caratteristiche medie del fluido. La soluzione numerica di quest'ultimo approccio risulta più agevole dal punto di vista computazionale.
Il processo di mediazione comporta l'introduzione di proprietà medie del fluido, ma in molti problemi di natura ingegneristica è sufficiente conoscere le proprietà medie piuttosto che l'intera evoluzione dettagliata.
Per quanto riguarda gli aspetti legati alla turbolenza e all'interazione microscopica tra le fasi, si utilizzano dei modelli che ricorrono a grandezze già note nel sistema, come si vedrà nei paragrafi 2.3.2 e 2.3.4.

2.2.2 Formulazione euleriana locale e istantanea

Se si considera una regione V, fissa nello spazio e condivisa tra N fasi, è possibile ottenere le equazioni di bilancio di una generica variabile Ψ, vettoriale o scalare. È possibile indicare V come unione di N sottoregioni, ognuna delle quali occupata da una fase k differente:

$$V = \bigcup_{k=1}^{N} V_k(t) \tag{2.1}$$

dove $k = 1, ..., N$.
L'equazione di bilancio della variabile Ψ per il volume di controllo V può essere scritta come segue, considerando l'interfaccia fra le fasi come una superficie geometrica senza massa:

$$\begin{aligned}\sum_{k=1}^{N}\left(\frac{d}{dt}\int_{V_k(t)}\rho_k\Psi_k dV\right) = \\ = -\sum_{k=1}^{N}\left(\int_{A_k(t)}\rho_k\Psi_k(\vec{u}_k\cdot\vec{n}_k)dA\right) - \sum_{k=1}^{N}\left(\int_{A_k(t)}(\vec{J}_k\cdot\vec{n}_k)dA\right) + \\ + \frac{1}{2}\sum_{k=1}^{N}\sum_{j=1}^{N}(1-\delta_{jk})\int_{A_{I,jk}(t)}\Phi_{I,jk}dA + \sum_{k=1}^{N}\int_{V_k(t)}\rho_k\Phi_k dV\end{aligned} \tag{2.2}$$

dove $A_{I,jk}(t)$ rappresenta l'interfaccia tra la $j-esima$ e la $k-esima$ fase, $A_k(t)$ è la superficie del volume $V_k(t)$, ρ_k è la densità della $k-esima$ fase, Ψ_k è una proprietà estensiva della $k-esima$ fase, $\vec{n}_k$ rappresenta il versore normale uscente dall'interfaccia del volume occupato dalla fase k, $\vec{u}_k$ è la velocità della fase k, $\vec{J}_k$ è il flusso molecolare uscente dalla fase k, Φ_k è il termine sorgente, $\Phi_{I,jk}$ è il termine sorgente dell'interfaccia e δ_{jk} rappresenta il delta di Kronecker.
Nell'equazione 2.2 il primo termine rappresenta l'accumulo, il primo termine a destra dell'uguale è il termine convettivo, il secondo quello diffusivo, mentre gli ultimi due sono termini sorgenti relativi all'interfaccia e al volume di controllo.
Si può descrivere la posizione dell'interfaccia $A_{I,jk}(t)$ con il vettore:

$$\vec{r}_{I,jk} = \vec{r}_{I,jk}(x(\zeta,\eta,t), y(\zeta,\eta,t), z(\zeta,\eta,t)) \tag{2.3}$$

e la velocità del punto sulla superficie di coordinate (ζ, η), può essere definita come:

$$\vec{u}_{I,jk} = \left(\frac{\partial\vec{r}_{I,jk}}{\partial t}\right)_{\zeta,\eta=cost} \tag{2.4}$$

Il termine di accumulo, relativo all'equazione 2.2, può essere rielaborato utilizzando il teorema di Leibniz, scomponendolo in un integrale di volume e di superficie.

$$\begin{aligned}\frac{d}{dt}\int_{V_k(t)}\rho_k\Psi_k dV = \int_{V_k(t)}\frac{\partial\rho_k\Psi_k}{\partial t}dV + \sum_{j=1}^{N}(1-\delta_{jk})\int_{A_{I,jk}(t)}\rho_k\Psi_k\vec{u}_{I,jk}\cdot\vec{n}_{kj}dA + \\ + \int_{A_k(t)}\rho_k\Psi_k\vec{u}_{I,k}\cdot\vec{n}_k dA = \\ = \int_{V_k(t)}\frac{\partial\rho_k\Psi_k}{\partial t}dV + \sum_{j=1}^{N}(1-\delta_{jk})\int_{A_{I,jk}(t)}\rho_k\Psi_k\vec{u}_{I,jk}\cdot\vec{n}_{kj}dA\end{aligned} \tag{2.5}$$

L'integrale sulla superficie $A_k(t)$ è stato eliminato poichè la velocità in un generico punto della superficie è nulla, dal momento che si è assunto un volume di controllo fisso nello spazio.
Il termine convettivo e diffusivo possono essere riscritti come somma di un integrale di volume e superficie, utilizzando il teorema di Gauss:

$$\int_{A_k(t)} \rho_k \Psi_k \vec{u}_k \cdot \vec{n}_k dA = \int_{V_k(t)} \nabla \cdot (\rho_k \Psi_k \vec{u}_k) dV - \sum_{j=1}^{N} (1 - \delta_{jk}) \int_{A_{I,kj}(t)} \rho_k \Psi_k \vec{u}_k \cdot \vec{n}_{kj} dA \quad (2.6)$$

$$\int_{A_k(t)} \vec{J}_k \cdot \vec{n}_k dA = \int_{V_k(t)} \nabla \cdot \vec{J}_k dV - \sum_{j=1}^{N} (1 - \delta_{jk}) \int_{A_{I,kj}(t)} \vec{J}_k \cdot \vec{n}_{kj} dA \quad (2.7)$$

Sostituendo tutte le espressioni ottenute nell'equazione 2.2, si ricava:

$$\sum_{k=1}^{N} \int_{V_k(t)} \left[\frac{\partial \rho_k \Psi_k}{\partial t} + \nabla \cdot (\rho_k \Psi_k \vec{u}_k) + \nabla \cdot \vec{J}_k - \rho_k \Phi_k \right] dV + \\ - \sum_{k=1}^{N} \sum_{j=1}^{N} (1 - \delta_{jk}) \int_{A_{I,kj}(t)} \left[\dot{m}_{I,kj} \Psi_k + \vec{J}_k \cdot \vec{n}_{kj} + \frac{1}{2} \Phi_{I,kj} \right] dA = 0 \quad (2.8)$$

dove il termine $\dot{m}_{I,kj}$ rappresenta il trasferimento di massa nell'unità di tempo e di superficie dalla fase $k - esima$ alla fase $j - esima$ attraverso l'interfaccia, e può essere scritto come segue:

$$\dot{m}_{I,kj} = \rho_k (\vec{u}_k + \vec{u}_{I,kj}) \cdot \vec{n}_{I,kj}. \quad (2.9)$$

L'equazione 2.8 deve essere soddisfatta per qualsiasi $V_k(t)$ e $A_{I,kj}(t)$, quindi è possibile scrivere l'equazione di conservazione locale ed istantanea come:

$$\frac{\partial \rho_k \Psi_k}{\partial t} + \nabla \cdot (\rho_k \Psi_k \vec{u}_k) + \nabla \cdot \vec{J}_k - \rho_k \Phi_k = 0 \quad (2.10)$$

La condizione di salto all'interfaccia è rappresentata dalla seguente equazione:

$$\dot{m}_{I,kj} \Psi_k + \dot{m}_{I,jk} \Psi_k + \vec{J}_k \cdot \vec{n}_{I,kj} + \vec{J}_j \cdot \vec{n}_{I,jk} = \Phi_{I,jk} \quad (2.11)$$

Le equazioni 2.10 e 2.11 hanno validità generale per una qualsiasi grandezza estensiva scalare o vettoriale e da esse possono essere ricavate le equazioni di conservazione istantanee per massa, quantità di moto ed energia, sostituendo i valori indicati in tabella 2.1.

TIPO DI BILANCIO	Ψ_k	$\vec{J}_k$	Φ_k	$\Phi_{I,kj}$
Massa	1	0	0	0
Quantità di moto	$\vec{u}_k$	- $\overline{\overline{T}}$	$\vec{b}_k$	- $\vec{m}^{\sigma}_{jk}$
Energia	E_k	$\vec{q}_k$ - $\overline{\overline{T}}\cdot\vec{u}_k$	$\vec{b}_k\cdot\vec{u}_k + \vec{Q}_k$	- $\vec{\epsilon}^{\sigma}_{jk}$

Tabella 2.1: Valori di Ψ_k, $\vec{J}_k$, Φ_k e $\Phi_{I,kj}$ per le equazioni di bilancio e le condizioni di salto all'interfaccia.

L'equazione di conservazione della massa e la relativa condizione di salto all'interfaccia possono essere espresse come segue:

$$\frac{\partial \rho_k}{\partial t} + \nabla\cdot(\rho_k\vec{u}_k) = 0 \tag{2.12}$$

$$\dot{m}_{I,kj} + \dot{m}_{I,jk} = 0 \tag{2.13}$$

L'equazione di bilancio della quantità di moto e la condizione di salto all'interfaccia si possono scrivere come:

$$\frac{\partial \rho_k\vec{u}_k}{\partial t} + \nabla\cdot(\rho_k\vec{u}_k\vec{u}_k) = \nabla\cdot\overline{\overline{T}}_k + \rho_k\cdot\vec{b}_k \tag{2.14}$$

$$\dot{m}_{I,kj}\vec{u}_k + \dot{m}_{I,jk}\vec{u}_j + \overline{\overline{T}}_k\cdot\vec{n}_{I,kj} + \overline{\overline{T}}_j\cdot\vec{n}_{I,jk} = \vec{m}^{\sigma}_{jk} \tag{2.15}$$

dove $\overline{\overline{T}}$ è il tensore degli sforzi, $\vec{b}_k$ rappresenta le forze che agiscono sul corpo e $\vec{m}^{\sigma}_{jk}$ è la trazione superficiale definita come:

$$\vec{m}^{\sigma}_{jk} = 2H_{jk}\sigma\vec{n}_{I,jk} - \nabla_{I,jk}\sigma \tag{2.16}$$

in cui σ è la tensione superficiale, H_{jk} rappresenta la curvatura media dell'interfaccia e $\nabla_{I,jk}\sigma$ il gradiente della tensione superficiale sull'interfaccia tra le fasi $j-esima$ e $k-esima$.
L'equazione di bilancio dell'energia e la condizione di salto si possono scrivere come segue:

$$\frac{\partial \rho_k E_k}{\partial t} + \nabla\cdot(\rho_k E_k\vec{u}_k) = -\nabla\cdot(\vec{q}_k - \overline{\overline{T}}_k\cdot\vec{u}_k) + \rho_k(\vec{b}_k\cdot\vec{u}_k + Q_k) \tag{2.17}$$

$$\dot{m}_{I,kj}E_k + \dot{m}_{I,jk}E_j + (\vec{q}_k - \overline{\overline{T}}_k\cdot\vec{u}_k)\cdot\vec{n}_{I,kj} + (\vec{q}_j - \overline{\overline{T}}_j\cdot\vec{u}_j)\cdot\vec{n}_{I,jk} = \epsilon^{\sigma}_{jk} \tag{2.18}$$

dove $\vec{q}_k$ è il flusso di calore, Q_k è il termine sorgente del calore per unità di massa e ϵ^{σ}_{jk} rappresenta l'energia superficiale associata all'interfaccia tra la fase j e la fase k e indicata come:

$$\epsilon^{\sigma}_{jk} = \nabla_{I,jk}\cdot\vec{q}_{I,jk} - \nabla_{I,jk}\cdot(\sigma_{jk}\vec{u}_{I,jk}) \tag{2.19}$$

dove $\vec{u}_{I,jk}$ è la componente tangenziale della velocità superficiale e $\vec{q}_{I,jk}$ rappresenta il flusso di calore scambiato tra la fase j e la fase k.

2.2.3 Tecniche di mediazione

Come detto in precedenza, nei casi in cui non sia richiesto conoscere i dettagli della dinamica del fluido, interfaccia tra le fasi compresa, si utilizzano le tecniche di mediazione che permettono di ricavare delle equazioni di conservazione mediate.
Le tecniche di mediazione più comuni sono quella temporale, volumica e di insieme. Prima però occorre distinguere tra le diverse fasi del sistema durante il processo di mediazione, in modo tale che i contributi dell'equazione mediata provenienti da una certa fase abbiano origine solamente da regioni che contengono la fase in questione.
Si definisce pertanto l'indicatore di fase X_k per la fase k come $X_k = (\vec{x}, t; \phi)$, se il vettore posizione $\vec{x}$ appartiene ad un qualsiasi processo ϕ associato alla fase k l'indicatore è pari ad 1, altrimenti è pari a 0. X_k si presenta, quindi, come una funzione a gradino.
La derivata materiale dell'indicatore di fase è sempre nulla e si può indicare come segue:

$$\frac{\partial X_k}{\partial t} + \vec{u}_{I,jk} \cdot \nabla X_k = 0 \tag{2.20}$$

Il gradiente dell'indicatore di fase viene scritto come:

$$\nabla X_k = \left(\frac{\partial X_k}{\partial n} \right) \vec{n}_{I,kj} = \delta(\vec{x} - \vec{x}_{I,kj}) \vec{n}_{I,kj} \tag{2.21}$$

dove $\delta(\vec{x} - \vec{x}_{I,kj})$ rappresenta la funzione delta di Dirac associata all'interfaccia tra le fasi $k - esima$ e $j - esima$.
La media nel tempo della variabile $f(\vec{x}, t; \phi)$, rappresentante una generica proprietà del fluido, si può scrivere come:

$$\langle f(\vec{x}, t; \phi) \rangle^T = \frac{1}{T} \int_{t-\frac{T}{2}}^{t+\frac{T}{2}} f(\vec{x}, t; \phi) d\tau \tag{2.22}$$

dove T indica il dominio temporale.
La media volumica, invece, si esprime come:

$$\langle f(\vec{x}, t; \phi) \rangle^V = \frac{1}{V} \int_V f(\vec{x}, t; \phi) dV_\eta \tag{2.23}$$

Infine la media di insieme viene indicata come segue:

$$\langle f(\vec{x}, t; \phi) \rangle^\varepsilon = \int_\varepsilon f(\vec{x}, t; \phi) dm(\phi) \tag{2.24}$$

dove $dm(\phi)$ rappresenta la densità di probabilità nell'insieme di tutti gli eventi ε.
Tutte le tecniche di mediazione soddisfano le regole di mediazione di Reynolds, che possono essere così riassunte:

$$\langle f + g \rangle = \langle f \rangle + \langle g \rangle \tag{2.25}$$

$$\langle \langle f \rangle g \rangle = \langle f \rangle \langle g \rangle \tag{2.26}$$

$$\langle Costante \rangle = Costante \tag{2.27}$$

$$\langle \frac{\partial f}{\partial t} \rangle = \frac{\partial \langle f \rangle}{\partial t} \tag{2.28}$$

$$\langle \nabla f \rangle = \nabla \langle f \rangle \tag{2.29}$$

$$\langle \nabla \cdot f \rangle = \nabla \cdot \langle f \rangle \tag{2.30}$$

Concettualmente la media d'insieme è la procedura di mediazione migliore dal punto di vista matematico, poiché non dipende dal periodo T o dal volume V, ma dal momento che tutte le procedure di mediazione portano alle stesse equazioni finali mediate, in questa trattazione si è scelto di ricorrere alla media volumica.
Nel metodo numerico dei volumi finiti, facendo riferimento alla media volumica, i valori delle variabili in ogni cella vengono mediati su tutto il volume della cella stessa.
L'equazione di bilancio mediata è ottenuta moltiplicando l'equazione (2.10) di bilancio locale per l'indicatore di fase e mediando per il volume:

$$\frac{\partial \langle X_k \rho_k \Psi_k \rangle}{\partial t} + \nabla \cdot \langle X_k \rho_k \vec{u}_k \rangle = -\langle X_k \vec{J}_k \rangle + \langle X_k \rho_k \Phi_k \rangle + \\ - \frac{1}{V} \sum_{j=1}^{N} (1 - \delta_{jk}) \int_{A_{I,kj}} [\dot{m}_{I,kj} \Psi_k + \vec{J}_k \cdot \vec{n}_{I,kj}] dA \tag{2.31}$$

L'equazione mediata per la condizione di salto all'interfaccia si può ricavare moltiplicando l'equazione 2.11 per il gradiente dell'indicatore di fase e mediando secondo il volume:

$$\frac{1}{V} \int_V [\dot{m}_{I,kj} \Psi_k + \dot{m}_{I,jk} \Psi_j + \vec{J}_k \cdot \vec{n}_{I,kj} + \vec{J}_j \cdot \vec{n}_{I,jk}] \frac{\partial X_k}{\partial \vec{n}_{I,kj}} dV = \frac{1}{V} \int_V \Phi_{I,jk} \frac{\partial X_k}{\partial \vec{n}_{I,kj}} dV \tag{2.32}$$

Operando le opportune semplificazioni, l'equazione precedente può essere riscritta come segue:

$$\frac{1}{V} \int_V [\dot{m}_{I,kj} \Psi_k + \dot{m}_{I,jk} \Psi_j + \vec{J}_k \cdot \vec{n}_{I,kj} + \vec{J}_j \cdot \vec{n}_{I,jk}] dA = \frac{1}{V} \int_V \Phi_{I,jk} dA \tag{2.33}$$

2.2.4 Modello Multifluido

Quando non è necessaria una descrizione dettagliata della dinamica dell'interfaccia, il processo di mediazione può essere effettuato su un volume più grande di quello della fase dispersa. In questo modo però si perdono tutte le informazioni di scala inferiore rispetto a quella su cui si effettua la mediazione.
Per tenere conto di questo problema, si introducono dei sottomodelli, ottenuti per via empirica, numerica o analitica.
Le equazioni che descrivono il modello multifluido, possono essere scritte mediante le seguenti medie:

1. Media fasica

$$\Psi_k = \frac{\langle X_k \Psi_k \rangle}{\alpha_k} \tag{2.34}$$

dove α_k è la frazione volumica della fase k, definita come la media volumica dell'indicatore di fase, per la quale si assume che:

$$\sum_{k=1}^{N} \alpha_k = 1 \tag{2.35}$$

2. Media di Favre

$$\hat{\Psi}_k = \frac{X_k \rho_k \Psi_k}{\alpha_k \rho_k} \tag{2.36}$$

Si definisce, inoltre, l'area interfacciale per unità di volume come:

$$A_k = -\langle \vec{n}_k \cdot \nabla X_k \rangle \tag{2.37}$$

dove $\vec{n}_k$ rappresenta il versore normale alla fase k.
L'equazione di bilancio mediata per una generica variabile può essere espressa come:

$$\frac{\partial \alpha_k \bar{\rho}_k \hat{\Psi}_k}{\partial t} + \nabla \cdot (\alpha_k \bar{\rho}_k \hat{\Psi}_k \hat{\vec{u}}_k) = -\nabla \cdot [\alpha_k (\hat{\vec{J}}_k + \hat{\vec{J}}_k^T)] + \langle \alpha_k \bar{\rho} \hat{\Phi}_k \rangle + \\ - \frac{1}{V} \sum_{j=1}^{N} (1 - \delta_{jk}) \int_{A_{I,kj}} [\dot{m}_{I,kj} \Psi_k + \vec{J}_k \cdot \vec{n}_{I,kj}] dA \tag{2.38}$$

dove $\vec{J}_k^T$ deriva dal termine convettivo in seguito all'operazione di mediazione.
Contiene tutte le informazioni sulla turbolenza relative ad una scala inferiore rispetto a quella risolta e può essere indicato come segue:

$$\vec{J}_k^T = \bar{\rho}_k (\widehat{\Psi_k \vec{u}_k} - \hat{\Psi}_k \hat{\vec{u}}_k) \tag{2.39}$$

È possibile ricavare le equazioni di conservazione della massa, della quantità di moto e dell'energia, sostituendo i rispettivi valori mostrati in tabella 2.1.
Per l'equazione di conservazione della massa, è possibile ricavare:

$$\frac{\partial \alpha_k \bar{\rho}_k}{\partial t} + \nabla \cdot (\alpha_k \bar{\rho}_k \hat{\vec{u}}_k) = \Gamma_k \tag{2.40}$$

È valida la seguente relazione:

$$\sum_{k=1}^{N} \Gamma_k = 0 \tag{2.41}$$

dove Γ_k rappresenta il termine di trasporto di massa tra le fasi e si può scrivere come:

$$\Gamma_k = -\frac{1}{V} \sum_{j=1}^{N} (1 - \delta_{jk}) \int_{A_{I,kj}} \dot{m}_{I,kj} dA \tag{2.42}$$

La condizione di salto all'interfaccia viene espressa dalla seguente relazione:

$$\frac{1}{V}\int_{A_{I,kj}}[\dot{m}_{I,kj}+\dot{m}_{I,jk}]dA=0 \tag{2.43}$$

Sostituendo i valori delle variabili della seconda riga in tabella 2.1, è possibile ricavare l'equazione di bilancio della quantità di moto:

$$\frac{\partial\alpha_k\bar{\rho}_k\hat{\vec{u}}_k}{\partial t}+\nabla\cdot(\alpha_k\bar{\rho}_k\hat{\vec{u}}_k\hat{\vec{u}}_k) = \nabla\cdot[\alpha_k(\hat{\overline{\overline{T}}}_k+\overline{\overline{T}}_k^T)]+\alpha_k\bar{\rho}_k\vec{g}+\Gamma_k\hat{\vec{u}}_k+\vec{M}_{I,k} \tag{2.44}$$

in cui $\overline{\overline{T}}_k^T$ è il tensore degli sforzi residui dovuti alla turbolenza alle scale minori di quella risolta.
In questo caso è valida la seguente espressione:

$$\sum_{k=1}^{N}[\Gamma_k\hat{\vec{u}}_k+\vec{M}_{I,k}]=\frac{1}{2}\sum_{k=1}^{N}\sum_{j=1}^{N}\frac{1}{V}(1-\delta_{kj})\int_{A_{I,kj}}\vec{m}_{jk}^{\sigma}dA=\vec{M}_{\sigma} \tag{2.45}$$

dove $\vec{M}_{I,k}$ rappresenta il termine che tiene conto di tutte le forze interfacciali presenti e viene indicato come:

$$\vec{M}_{I,k}=-\frac{1}{V}\sum_{j=1}^{N}(1-\delta_{jk})\int_{A_{I,kj}}[\dot{m}_{I,kj}\vec{u}_k-\overline{\overline{T}}_k\cdot\vec{n}_{I,kj}]dA \tag{2.46}$$

La condizione di salto all'interfaccia viene descritta dalla seguente equazione:

$$\frac{1}{V}\int_{A_{I,kj}}[\dot{m}_{I,kj}\vec{u}_k+\dot{m}_{I,jk}\vec{u}_j-\overline{\overline{T}}_k\cdot\vec{n}_{I,kj}-\overline{\overline{T}}_j\cdot\vec{n}_{I,jk}]dA=-\frac{1}{V}\int_{A_{I,kj}}\vec{m}_{jk}^{\sigma}dA \tag{2.47}$$

Infine utilizzando i valori dell'ultima riga in tabella 2.1, si ottiene l'equazione mediata dell'energia espressa tramite l'entalpia come variabile dipendente:

$$\begin{aligned}\frac{\partial\alpha_k\bar{\rho}_k\hat{h}_k}{\partial t}+\nabla\cdot(\alpha_k\bar{\rho}_k\hat{\vec{u}}_k\hat{h}_k) &= -\nabla\cdot\left(\alpha_k(\hat{\bar{q}}_k+\hat{\bar{q}}_k^T)\right)+\\ &+\nabla\cdot\left(\alpha_k(\hat{\overline{\overline{T}}}_k\cdot\hat{\vec{u}}_k)+(\hat{\overline{\overline{T}}}_k\cdot\hat{\vec{u}}_k)^T\right)+\alpha_k\bar{\rho}_kS_k+\hat{h}_{kj}\cdot\Gamma_k+E_k\end{aligned} \tag{2.48}$$

Il primo termine a destra dell'uguale definisce il trasporto di calore nella fase k originato dai gradienti di temperatura.
Il secondo rappresenta la dissipazione dovuta agli sforzi di taglio nella fase k. Infine gli ultimi tre termini rappresentano rispettivamente la generazione di energia nella fase k per reazione chimica, il trasporto di energia dovuto al trasferimento di massa tra le fasi e il trasporto di calore dalle altre fasi verso la fase k.

2.3 Chiusura del modello

Le equazioni mediate, descritte nel paragrafo precedente, non sono direttamente risolubili per la presenza di alcuni termini, che possono essere così riassunti:

1. autointerazione ($\overline{\overline{T}}_k$);
2. interazione tra le fasi (Γ_k, $\overrightarrow{M}_{I,k}$,$\vec{M}_\sigma$);
3. turbolenza ($\overline{\overline{T}}_k^T$).

Per la chiusura del modello è quindi necessario definire lo stato termodinamico del sistema, per mezzo di relazioni tra le diverse variabili termodinamiche.
Dopo aver esplicitato le condizioni iniziali e al contorno, e stabilite le relazioni di chiusura, è possibile risolvere numericamente le equazioni che descrivono il sistema in esame.
È bene sottolineare che le equazioni di chiusura ricercate devono rispettare le condizioni utilizzate nella derivazione delle leggi costitutive:

- principio di equipresenza delle variabili;
- indipendenza dal sistema di riferimento;
- esistenza e unicità della soluzione alle equazioni;
- determinismo;
- validità della seconda legge della termodinamica.

Di seguito vengono trattate le leggi di chiusura utilizzate per un sistema gas-liquido in cui la fase dispersa è costituita da un insieme di bolle di gas.

2.3.1 Auto-interazione

Il termine di auto-interazione è rappresentato dal tensore degli sforzi viscosi relativo alla fase k:

$$\overline{\overline{T}}_k = -p_k\overline{\overline{I}} + \overline{\overline{\tau}}_k \tag{2.49}$$

Come è possibile notare dall'equazione 2.49, $\overline{\overline{T}}_k$ viene suddiviso in una componente di pressione e in una componente di sforzo di taglio.
Quest'ultima viene indicata secondo la relazione tensione-sforzo di Newton:

$$\overline{\overline{\tau}}_k = \xi_k(\nabla\cdot\hat{\vec{u}}_k)\overline{\overline{I}} + 2\mu_k[\overline{\overline{S}}_k - \frac{1}{3}\nabla\cdot\hat{\vec{u}}_k\overline{\overline{I}}] \tag{2.50}$$

Il termine $\overline{\overline{S}}_k$ è il tensore della velocità di tensione, ed è definito come segue:

$$\overline{\overline{S}}_k = \frac{1}{2}\left(\nabla\hat{\vec{u}}_k + (\nabla\hat{\vec{u}}_k)^T\right) \tag{2.51}$$

Si assume, inoltre, che la viscosità di massa ξ_k sia nulla per tutte le fasi e che la viscosità dinamica μ_k sia pari ad un valore costante per la fase continua e per la fase dispersa.

2.3.2 Interazione tra le fasi

Il termine di scambio di massa Γ_k può essere assunto pari a zero, poichè nel processo in esame non è presente alcun cambiamento di stato, né trasferimento di materia.
Il secondo termine di interazione tra le fasi che necessita di un modello di chiusura è la forza interfacciale per unità di volume per la fase k, che può essere suddiviso in un termine che rappresenta le forze per unità di volume, in una pressione media interfacciale e in un termine che è funzione dello sforzo di taglio medio interfacciale.

$$\vec{M}_{I,k} = \vec{M}^{gd}_{I,k} + p_{I,k}\nabla\alpha_k - \bar{\bar{\tau}}_{I,k}\nabla\alpha_k \tag{2.52}$$

Riscrivendo i termini a destra dell'uguale nell'equazione di bilancio di quantità di moto 2.44, si ottiene:

$$\begin{aligned}&\nabla\cdot(\alpha_k\hat{\bar{\bar{T}}}_k) + \alpha_k\bar{\rho}_k\vec{g} + \Gamma_k\hat{\vec{u}}_k + \vec{M}_{I,k} = \\ &= -\alpha_k\nabla p_k + \nabla\cdot(\alpha_k\bar{\bar{\tau}}_k) + (p_{I,k} - p_k)\nabla\alpha_k - \tau_{I,k}\nabla\cdot\alpha_k + \alpha_k\bar{\rho}_k\vec{g} + \Gamma_k\hat{\vec{u}}_k + \vec{M}^{gd}_{I,k}\end{aligned} \tag{2.53}$$

Il termine $\vec{M}^{gd}_{I,k}$ rappresenta tutte le forze per unità di volume che agiscono all'interfaccia tra la fase dispersa e quella continua.
Per sistemi diluiti si può assumere che le forze interfacciali agiscano esclusivamente tra la fase primaria ($k = N$) e le fasi secondarie ($k = 1, ..., N-1$), dove la fase primaria corrisponde a quella liquida e le fasi secondarie a quelle gassose.
Il termine di forze interfacciali agenti sulla fase primaria si può esprimere come combinazione lineare di vari contributi [14]:

$$\vec{M}^{gd}_{I,N} = \sum_{k=1}^{N-1}(\vec{M}^{D}_{I,k} + \vec{M}^{L}_{I,k} + \vec{M}^{VM}_{I,k} + \vec{M}^{B}_{I,k} + \vec{M}^{TD}_{I,k}) \tag{2.54}$$

Per le fasi secondarie si può scrivere:

$$\vec{M}^{gd}_{I,k} = -(\vec{M}^{D}_{I,k} + \vec{M}^{L}_{I,k} + \vec{M}^{VM}_{I,k} + \vec{M}^{B}_{I,k} + \vec{M}^{TD}_{I,k}) \tag{2.55}$$

Nelle equazioni 2.54 e 2.55 vengono indicati rispettivamente i contributi dati dalla forza di drag, dalla risalita, dalla massa virtuale, dalla forza di Basset e dalla forza di dispersione turbolenta.
Il primo contributo è originato dal moto relativo (slip velocity) tra la fase secondaria e la fase primaria. La forza di risalita o di lift è la forza trasversale dovuta ai gradienti di velocità presenti nella fase continua, che assume più importanza quando le particelle sono grandi.
La forza di massa virtuale è originata dall'accelerazione della fase secondaria rispetto alla primaria, l'inerzia della massa della fase primaria esercita, quindi, una forza sulle particelle.
La forza di Basset è la forza viscosa dovuta all'accelerazione relativa tra la fase continua e quella dispersa, e allo sviluppo dello strato limite. La forza di dispersione turbolenta tiene conto delle forze originate dai moti dispersivi della turbolenza, che influiscono sulle forze agenti all'interfaccia.

L'unico contributo considerato in questa trattazione è quello relativo alla forza di drag che verrà discusso nel paragrafo successivo.
Come già accennato, i termini di interazione $\vec{M}_{I,k}$ si definiscono come forze per unità di volume:

$$\vec{M}_{I,k} = \frac{\alpha_k \vec{F}_{I,k}}{V_b} \tag{2.56}$$

dove $\vec{F}_{I,k}$ rappresenta una generica forza che agisce sulla particella.
Il volume della particella nel caso di bolle sferiche si scrive come:

$$V_b = \frac{\pi}{6} \bar{d}_{b,k}^3 \tag{2.57}$$

dove $\bar{d}_{b,k}$ è il diametro della particella.

2.3.3 Forza di trascinamento

La forza di trascinamento o di drag è una forza che si origina in seguito al moto relativo di un corpo sommerso che si muove rispetto al fluido circostante. Questo tipo di forza esiste anche nel caso di particelle fluide, come per il caso di una bolla di gas immersa in un liquido.
La forza di drag dipende sia dall'attrito superficiale dovuto agli sforzi di taglio alla superficie del corpo sommerso, sia dalla forma determinata dalla distribuzione non uniforme della pressione dovuta al moto. Si può esprimere secondo la legge:

$$\vec{F}_{I,k}^D = \frac{1}{2} \rho_N C_{D,k} A |\vec{u}_r| \vec{u}_r \tag{2.58}$$

dove $C_{D,k}$ è il coefficiente di drag, A la proiezione dell'area del corpo su una superficie normale alla velocità relativa fra i due fluidi e $\vec{u}_r$ la velocità relativa. Quest'ultima si può scrivere come:

$$\vec{u}_r = \vec{u}_k - \vec{u}_N \tag{2.59}$$

La proiezione dell'area per una bolla sferica è pari a:

$$A = \frac{\pi}{4} \bar{d}_{b,k}^2 \tag{2.60}$$

Il contributo dato dalla forza di drag nell'equazione 2.54 e 2.55, può essere ricavato sostituendo nell'equazione 2.56 le relazioni 2.58, 2.59 e 2.60:

$$\vec{M}_{I,k}^D = \frac{3}{4} \alpha_k \frac{C_{D,k} \rho_N}{\bar{d}_{b,k}} |\hat{\vec{u}}_k - \hat{\vec{u}}_N| (\hat{\vec{u}}_k - \hat{\vec{u}}_N) \tag{2.61}$$

Il coefficiente di drag è funzione del numero di Reynolds come mostrato in figura 2.1, e nel caso di particelle fluide, dipende dalla forma delle particelle.
Può essere stimato mediante l'utilizzo di correlazioni di natura empirica, tra cui quelle di Tomiyama, Schiller-Naumann o Ishii e Zuber, solamente per citarne qualcuna.
Nel caso in esame viene utilizzata la relazione di Schiller-Naumann [15], di seguito mostrata:

$$\begin{cases} C_D = \frac{24}{Re}(1 + 0.15Re^{0.687}) & Re < 1000 \\ C_D = 0.44 & Re > 1000 \end{cases} \tag{2.62}$$

dove il numero di Reynolds per sistemi con particelle è definito come:

$$Re = \frac{\rho_k \bar{d}_{b,k}}{\mu_N} |\vec{u}_r| \tag{2.63}$$

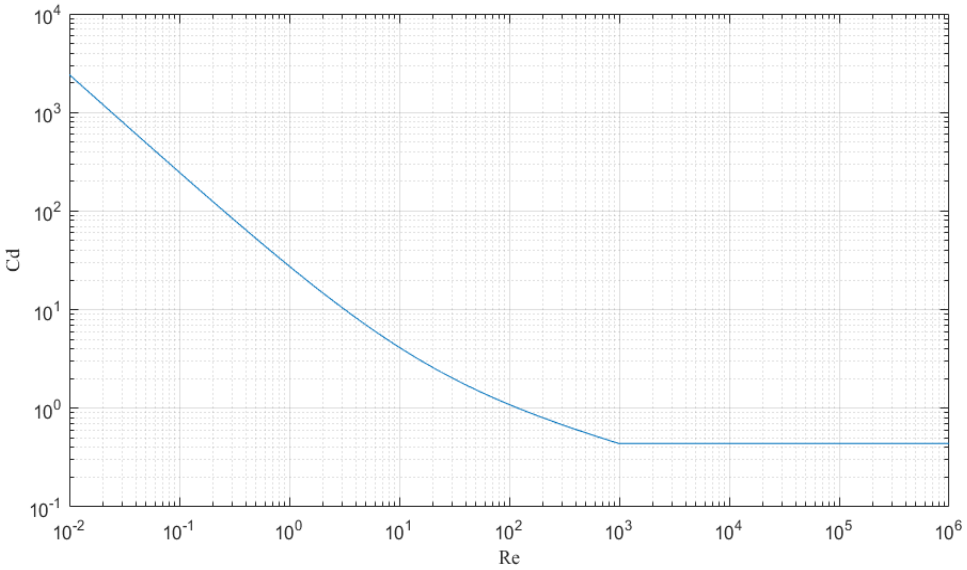

Figura 2.1: Andamento del coefficiente di drag al variare del numero di Reynolds per la relazione empirica nell'equazione 2.62.

2.3.4 Turbolenza

Il termine che tiene conto dell'effetto della turbolenza è il tensore degli sforzi di Reynols $\overline{\overline{T}}_k^T$. Nei sistemi multifase la modellazione della turbolenza diventa molto complicata a causa della natura del fenomeno. In presenza di un campo turbolento, i termini fluttuanti di velocità necessitano di un modello di chiusura.
Un approccio è rappresentato dalla risoluzione diretta delle equazioni di Navier-Stokes, noto come Direct Numerical Simulation (DNS), il suo utilizzo però è limitato al caso di scale spaziali e temporali ridotte, in modo tale da dover impiegare geometrie semplici e quindi griglie poco fitte.
Il metodo più diffuso è il cosidetto RANS (Reynolds Averaged Navier-Stokes), che si basa su una suddivisione del moto turbolento formata da un moto medio e da una sua fluttuazione nel tempo. Le equazioni vengono mediate in un certo intervallo di tempo, in modo tale che i tempi di calcolo siano notevolmente ridotti, poichè le scale del moto medio risultano essere notevolmente maggiori di quelle del moto turbolento. Richiedono l'utilizzo di ulteriori equazioni (ad esempio il modello k-ϵ) per la chiusura del problema.
Un altro approccio è rappresentato dal LES (Large Eddy Simulation), che si propone di operare un filtraggio delle equazioni di Navier-Stokes, di calcolare numericamente il comportamento delle scale turbolente più grandi, e di modellare le scale più piccole della

dimensione del filtro.
Esistono anche metodi che integrano sia il LES che il RANS, come il DES (Detached Eddy Simulation), in cui si utilizza il modello RANS per flussi vicino alle pareti e una procedura LES lontano dalle pareti.
Per i modelli di chiusura, tra quelli k-ϵ, i più utilizzati sono:

- mixture turbulence model;
- per phase turbulence model;
- dispersed turbulence model.

Il primo modello risolve le equazioni di trasporto di k ed ϵ riferite alla miscela delle fasi coinvolte, mentre il secondo risolve le stesse equazioni per tutte le fasi. Quest'ultimo è preferibile quando il trasferimento della turbolenza tra le diverse fasi è il meccanismo più importante. Il terzo modello, invece, viene utilizzato quando le fasi secondarie del sistema sono diluite nella fase primaria. In questo caso le interazioni tra le fasi secondarie vengono trascurate, perchè il processo dominante risulta essere la turbolenza della fase primaria.
In questa trattazione non viene analizzato nessun modello di chiusura della turbolenza, poichè il flusso del sistema in esame si trova in regime laminare.

2.4 Riassunto modelli con chiusura

2.4.1 Modello Two-Fluid

Per il modello Two-fluid con sole due fasi, l'equazione di conservazione della massa per la fase dispersa è espressa come:

$$\frac{\partial \alpha_1 \bar{\rho}_1}{\partial t} + \nabla \cdot (\alpha_1 \bar{\rho}_1 \hat{\vec{u}}_1) = 0 \tag{2.64}$$

In modo analogo si può ricavare la stessa espressione per la fase continua:

$$\frac{\partial \alpha_2 \bar{\rho}_2}{\partial t} + \nabla \cdot (\alpha_2 \bar{\rho}_2 \hat{\vec{u}}_2) = 0 \tag{2.65}$$

In alternativa si può ricavare la frazione volumica della fase 2, sapendo che:

$$\alpha_1 + \alpha_2 = 1 \tag{2.66}$$

Si noti come non ci sia nessun termine relativo al trasferimento di materia tra le fasi, come spiegato nel paragrafo 2.3.2.
Le equazioni di bilancio della quantità di moto per la fase dispersa e per quella continua sono rispettivamente:

$$\begin{aligned}\frac{\partial \alpha_1 \bar{\rho}_1 \hat{\vec{u}}_1}{\partial t} + \nabla \cdot (\alpha_1 \bar{\rho}_1 \hat{\vec{u}}_1 \hat{\vec{u}}_1) = \nabla \cdot (\alpha_1 \mu_1 (\nabla \hat{\vec{u}}_1 + \nabla \hat{\vec{u}}_1^T)) + \\ - \nabla \cdot (\alpha_1 \mu_1 \frac{1}{3} \nabla \hat{\vec{u}}_1 \bar{\bar{I}}) - \alpha_1 \nabla p + \alpha_1 \bar{\rho}_1 \vec{g} - \frac{3}{4} \bar{\rho}_2 \alpha_1 \frac{C_{D,1}}{\bar{d}_{b,1}} |\hat{\vec{u}}_1 - \hat{\vec{u}}_2| (\hat{\vec{u}}_1 - \hat{\vec{u}}_2)\end{aligned} \tag{2.67}$$

$$\frac{\partial \alpha_2 \bar{\rho}_2 \hat{\vec{u}}_2}{\partial t} + \nabla\cdot(\alpha_2 \bar{\rho}_2 \hat{\vec{u}}_2 \hat{\vec{u}}_2) = \nabla\cdot(\alpha_2 \mu_2 (\nabla \hat{\vec{u}}_2 + \nabla \hat{\vec{u}}_2^T)) +$$
$$- \nabla\cdot(\alpha_2 \mu_2 \frac{1}{3} \nabla \hat{\vec{u}}_2 \overline{\overline{I}}) - \alpha_2 \nabla p + \alpha_2 \bar{\rho}_2 \vec{g} + \frac{3}{4} \bar{\rho}_2 \alpha_1 \frac{C_{D,1}}{\bar{d}_{b,1}} |\hat{\vec{u}}_1 - \hat{\vec{u}}_2| (\hat{\vec{u}}_1 - \hat{\vec{u}}_2) \quad (2.68)$$

Si noti come le equazioni di bilancio di quantità di moto sono state scritte considerando solamente il contributo dato dalla forza di drag descritta nel paragrafo 2.3.3.

2.4.2 Modello Multi-Fluid

Il modello multifluido con le equazioni di chiusura può essere riassunto come di seguito. L'equazione di conservazione di massa per la fase continua si può scrivere come:

$$\frac{\partial \alpha_N \bar{\rho}_N}{\partial t} + \nabla\cdot(\alpha_N \bar{\rho}_N \hat{\vec{u}}_N) = 0 \quad (2.69)$$

Per le fasi disperse $k = 1, ..., N-1$:

$$\frac{\partial \alpha_k \bar{\rho}_k}{\partial t} + \nabla\cdot(\alpha_k \bar{\rho}_k \hat{\vec{u}}_k) = 0 \quad (2.70)$$

È valida la relazione:

$$\sum_{k=1}^{N} \alpha_k = 1 \quad (2.71)$$

L'equazione di bilancio della quantità di moto per la fase primaria si può scrivere come:

$$\frac{\partial \alpha_N \bar{\rho}_N \hat{\vec{u}}_N}{\partial t} + \nabla\cdot(\alpha_N \bar{\rho}_N \hat{\vec{u}}_N \hat{\vec{u}}_N) = \nabla\cdot(\alpha_N \mu_N (\nabla \hat{\vec{u}}_N + \nabla \hat{\vec{u}}_N^T)) - \nabla\cdot(\alpha_N \mu_N \frac{1}{3} \nabla \hat{\vec{u}}_N \overline{\overline{I}}) +$$
$$- \alpha_N \nabla p + \alpha_N \bar{\rho}_N \vec{g} + \sum_{k=1}^{N-1} \frac{3}{4} \bar{\rho}_N \alpha_k \frac{C_{D,k}}{\bar{d}_{b,k}} |\hat{\vec{u}}_k - \hat{\vec{u}}_N| (\hat{\vec{u}}_k - \hat{\vec{u}}_N) \quad (2.72)$$

Per le fasi secondarie ($k = 1, ..., N-1$):

$$\frac{\partial \alpha_k \bar{\rho}_k \hat{\vec{u}}_k}{\partial t} + \nabla\cdot(\alpha_k \bar{\rho}_k \hat{\vec{u}}_k \hat{\vec{u}}_k) = \nabla\cdot(\alpha_k \mu_k (\nabla \hat{\vec{u}}_k + \nabla \hat{\vec{u}}_k^T)) - \nabla\cdot(\alpha_k \mu_k \frac{1}{3} \nabla \hat{\vec{u}}_k \overline{\overline{I}}) +$$
$$- \alpha_k \nabla p + \alpha_k \bar{\rho}_k \vec{g} - \frac{3}{4} \bar{\rho}_N \alpha_k \frac{C_{D,k}}{\bar{d}_{b,k}} |\hat{\vec{u}}_k - \hat{\vec{u}}_N| (\hat{\vec{u}}_k - \hat{\vec{u}}_N) \quad (2.73)$$

Capitolo 3

Implementazione

I solutori impiegati in questo lavoro sono il `twoPhaseEulerFoam` e il `multiphaseEulerFoam`: entrambi utilizzano un approccio di tipo euleriano-euleriano, e fanno parte della versione 5 di OpenFOAM.
Il `twoPhaseEulerFoam` è un solutore per sistemi con due fasi fluide comprimibili in cui una è la fase dispersa, mentre il `multiphaseEulerFoam` considera la presenza di più fasi fluide incomprimibili. Entrambi i solutori includono i bilanci di energia per il trasferimento di calore, ma in questa tesi non verranno considerati, perché il sistema trattato è isotermo.
L'obiettivo del lavoro è verificare l'abilità del codice `multiphaseEulerFoam` nel simulare sistemi multifase con l'approccio multifluido, confrontandolo con il codice `twoPhaseEulerFoam`, già utilizzato per lo studio di questi sistemi.
Per questo motivo, in questo capitolo verranno descritte le equazioni implementate nei codici, facendo particolare attenzione alle differenze presenti tra i due solutori e alle modifiche apportate in entrambi i codici, per rendere il confronto coerente.

3.1 `TwoPhaseEulerFoam`

Di seguito vengono descritte le equazioni che governano il modello Two-fluid, riassunte nel paragrafo 2.4.1, nella forma implementata nel codice `twoPhaseEulerFoam`.

3.1.1 Equazione di trasporto della frazione volumica

L'equazione di conservazione della massa per la fase dispersa nel modello Two-fluid viene scritta come:

$$\frac{\partial \alpha_1 \bar{\rho}_1}{\partial t} + \nabla \cdot (\alpha_1 \bar{\rho}_1 \hat{\vec{u}}_1) = 0 \tag{3.1}$$

Scomponendo i vari termini dell'equazione 3.1, si ottiene:

$$\bar{\rho}_1 \frac{\partial \alpha_1}{\partial t} + \alpha_1 \frac{\partial \bar{\rho}_1}{\partial t} + \bar{\rho}_1 \nabla \cdot (\alpha_1 \hat{\vec{u}}_1) + \alpha_1 \hat{\vec{u}}_1 \cdot \nabla \bar{\rho}_1 = 0 \tag{3.2}$$

Raggruppando i termini in comune, è possibile ricavare:

$$\rho_1 \left(\frac{\partial \alpha_1}{\partial t} + \nabla \cdot (\alpha_1 \hat{\vec{u}}_1) \right) + \alpha_1 \left(\frac{\partial \bar{\rho}_1}{\partial t} + \hat{\vec{u}}_1 \cdot \nabla \bar{\rho}_1 \right) = 0 \tag{3.3}$$

Dividendo l'equazione 3.3 per la densità, si ottiene:

$$\frac{\partial \alpha_1}{\partial t} + \nabla \cdot (\alpha_1 \hat{\vec{u}}_1) = -\frac{\alpha_1}{\bar{\rho}_1} \left(\frac{\partial \bar{\rho}_1}{\partial t} + \hat{\vec{u}}_1 \cdot \nabla \bar{\rho}_1 \right) \tag{3.4}$$

Il termine fra parentesi a destra del segno di uguaglianza rappresenta la derivata sostanziale della densità, che può essere così definita:

$$\frac{\mathcal{D}\bar{\rho}_1}{\mathcal{D}t} = \frac{\partial \bar{\rho}_1}{\partial t} + \hat{\vec{u}}_1 \cdot \nabla \bar{\rho}_1 \tag{3.5}$$

L'equazione 3.4 può essere riscritta come segue:

$$\frac{\partial \alpha_1}{\partial t} + \nabla \cdot (\alpha_1 \hat{\vec{u}}_1) = -\frac{\alpha_1}{\bar{\rho}_1} \frac{\mathcal{D}\bar{\rho}_1}{\mathcal{D}t} \tag{3.6}$$

Si ricorda che sono valide le seguenti relazioni:

$$\hat{\vec{u}} = \alpha_1 \hat{\vec{u}}_1 + \alpha_2 \hat{\vec{u}}_2 \tag{3.7}$$

$$\hat{\vec{u}}_r = \hat{\vec{u}}_1 - \hat{\vec{u}}_2 \tag{3.8}$$

$$\alpha_1 + \alpha_2 = 1 \tag{3.9}$$

Sostituendo l'equazione 3.8, esplicitata per la variabile $\vec{u}_2$, nell'equazione 3.7 si ottiene:

$$\hat{\vec{u}}_1 = \hat{\vec{u}} + (1 - \alpha_1) \hat{\vec{u}}_r \tag{3.10}$$

Utilizzando la definizione di $\hat{\vec{u}}_1$, è possibile riscrivere l'equazione 3.6 come:

$$\frac{\partial \alpha_1}{\partial t} + \nabla \cdot (\alpha_1 \hat{\vec{u}}) + \nabla \cdot (\alpha_1 (1 - \alpha_1) \hat{\vec{u}}_r) = -\frac{\alpha_1}{\bar{\rho}_1} \frac{\mathcal{D}\bar{\rho}_1}{\mathcal{D}t} \tag{3.11}$$

In modo analogo è possibile ricavare l'equazione di trasporto della frazione volumica per la fase continua.
Le equazioni implementate nel codice sono l'equazione 3.11 per la fase dispersa e l'equazione 3.9 per la fase continua. Il motivo per cui si effettuano queste sostituzioni è dovuto a problemi di instabilità numerica, causati dalla limitatezza della frazione volumica.
Se si tenta di risolvere l'equazione di trasporto della frazione volumica della fase dispersa nella forma in cui è scritta l'equazione 3.1, applicando un'opportuno schema di discretizzazione, il vincolo per cui $\alpha_k \geq 0$ verrà rispettato, mentre il vincolo per cui $\alpha_k < 1$ potrebbe non esserlo.
Risolvendo l'equazione 3.11, invece, la frazione volumica della fase k è limitata ad entrambi gli estremi dell'intervallo. Infatti, scomponendo il primo termine di convezione dell'equazione 3.11, si ottiene:

$$\nabla \cdot (\alpha_1 \hat{\vec{u}}) = \alpha_1 \nabla \cdot \hat{\vec{u}} + \hat{\vec{u}} \cdot \nabla \alpha_1 \tag{3.12}$$

dove il primo termine è nullo, perché $\nabla \cdot \vec{u} = 0$ quando non vi è trasferimento di materia tra le fasi, e il secondo è un termine di trasporto che ne preserva l'ampiezza.
Il secondo termine di convezione è anch'esso limitato ($0 \leq \alpha_k \leq 1$), poiché diventa nullo per entrambi i limiti.
Il codice utilizzato per la risoluzione del bilancio di materia è riportato in appendice A.1.

3.1.2 Equazione di bilancio della quantità di moto

Le equazioni di bilancio della quantità di moto per la fase dispersa e per quella continua nel modello Two-fluid sono esplicitate nel paragrafo 2.4.2.
Nel codice l'equazione 2.67 per la fase dispersa viene implementata come segue:

```
U1Eqn =
(
   fvm::ddt(alpha1, rho1, U1)
 + fvm::div(alphaRhoPhi1, U1)
 - fvm::Sp(contErr1, U1)
 + phase1.turbulence().divDevRhoReff(U1)
 ==
 - Vm *(fvm::ddt(U1) + fvm::div(phi1, U1)
     - fvm::Sp(fvc::div(phi1), U1)- DDtU2)
 + fvOptions(alpha1, rho1, U1)
);

U1Eqn += fvm::Sp(Kd, U1);
```

Il primo termine rappresenta l'accumulo, il secondo è il termine convettivo, il terzo tiene conto degli errori di continuità, e il quarto rappresenta il tensore degli sforzi viscosi molecolari e turbolenti.
A destra del segno di uguaglianza, il primo termine tiene conto della forza di massa virtuale che in questo lavoro non viene considerata, mentre il secondo la presenza di termini sorgente, in questo caso assenti.
Poiché, come visto nel capitolo precedente, la forza di trascinamento è funzione della velocità relativa `Ur` = `U1` − `U2`, questo termine viene suddiviso in una parte esplicita, indipendente dal valore di velocità `U1`, ed una implicita, funzione di `U1`.
Come si può notare, nel codice viene aggiunta solo la parte implicita (`Kd*U1`), mentre la parte esplicita (`-Kd*U2`) verrà inserita direttamente nell'equazione della pressione, come descritto nel paragrafo successivo.
Il termine `Kd` è il prodotto fra `Ki` e il modulo della velocità relativa `Ur`, dove `Ki` è definito come:

```
Ki =
 {
   0.75
   *CdRe()
   *swarmCorrection_Cs()
   *pair_.continuous().rho()
   *pair_.continuous().nu()
   /sqr(pair_.dispersed().d());
 }
```

Il coefficiente di `swarmCorrection` permette di correggere la forza di trascinamento, considerando che gli sciami di bolle si comportano diversamente dalle singole bolle, e in questo caso è impostato pari a 1.
Il termine `CdRe` rappresenta il coefficiente di drag, espresso dalla relazione di Schiller-Naumann mostrata nell'equazione 2.62, moltiplicato per il massimo fra il numero di Reynolds e il `residualRe`, come mostrato in appendice A.4. Il `residualRe` è un valore

molto piccolo che viene utilizzato quando il numero di Reynolds tende a zero, per evitare una suddivisione per zero all'interno delle equazioni.
Il termine `Ki` viene moltiplicato successivamente per il massimo fra la frazione volumica della fase dispersa e il `residualAlpha`, che è un valore molto piccolo che permettere di stabilizzare numericamente l'algoritmo iterativo di soluzione quando la frazione volumica tende a zero.

```
K=
{
 max(pair_.dispersed(), pair_.dispersed().residualAlpha())*Ki();
}
```

È semplice far notare come l'implementazione delle equazioni della teoria sia coerente in `twoPhaseEulerFoam`. Riassumendo i passaggi descritti, è possibile definire:

$$K = \frac{3}{4}\alpha_1 C_D Re \frac{\bar{\rho}_2 \nu_2}{\bar{d}_{b,1}^2} \tag{3.13}$$

Sostituendo la definizione del numero di Reynolds (eq. 2.63), si ottiene:

$$K = \frac{3}{4}\alpha_1 C_D \frac{\bar{d}_{b,1}|\hat{\vec{u}}_r|}{\nu_2} \frac{\bar{\rho}_2 \nu_2}{\bar{d}_{b,1}^2} \tag{3.14}$$

Operando le opportune semplificazioni, è possibile ricavare la stessa equazione 2.61 descritta nella teoria dei modelli euleriani:

$$K = \frac{3}{4}\alpha_1 C_D \frac{\bar{\rho}_2}{\bar{d}_{b,1}}|\hat{\vec{u}}_r| \tag{3.15}$$

Anche per la fase continua, come mostrato in appendice A, valgono le stesse considerazioni.

3.1.3 Algoritmo di accoppiamento pressione e velocità

Dopo aver descritto le equazioni di bilancio per la quantità di moto, è necessario definire l'equazione per calcolare la pressione.
Per derivare l'equazione della pressione, vengono esplicitate le velocità, espresse nel bilancio della quantità di moto, mediante la seguente formulazione:

$$\vec{u}_1 = \frac{H_1}{A_1} - \frac{\nabla p}{A_1 \bar{\rho}_1} \tag{3.16}$$

$$\vec{u}_2 = \frac{H_2}{A_2} - \frac{\nabla p}{A_2 \bar{\rho}_2} \tag{3.17}$$

dove A_1 e A_2 rappresentano le matrici dei coefficienti della velocità della fase dispersa e della fase continua, e H_1 e H_2 sono le matrici dei termini noti, all'infuori del termine relativo al gradiente di pressione.
Come già accennato nel paragrafo precedente, all'interno della matrice dei termini noti vengono aggiunti i termini relativi alla parte esplicita della forza di drag, come mostrato in appendice A.5.
Le equazioni per i flussi volumetrici alle facce sono create interpolando le equazioni 3.16

e 3.17:

$$\phi_1 = \phi_1^* - \left(\frac{1}{A_1\bar{\rho}_1}\right)_f S_f \nabla_f^{\perp} p \tag{3.18}$$

$$\phi_2 = \phi_2^* - \left(\frac{1}{A_2\bar{\rho}_2}\right)_f S_f \nabla_f^{\perp} p \tag{3.19}$$

dove i termini fra parentesi corrispondono ad un'operazione di interpolazione alle facce usando il metodo di Rhie-Chow, e ϕ_1^* e ϕ_2^* sono le predizioni dei flussi per la fase dispersa e per la fase continua, definite come:

$$\phi_1^* = \left(\frac{H_1}{A_1}\right)_f \cdot S_f \tag{3.20}$$

$$\phi_2^* = \left(\frac{H_2}{A_2}\right)_f \cdot S_f \tag{3.21}$$

Il flusso volumetrico totale alle facce è indicato come:

$$\phi = \alpha_{1,f}\phi_1 + \alpha_{2,f}\phi_2 \tag{3.22}$$

Il vincolo di continuità è:

$$\nabla\cdot\phi = -\frac{\alpha_1}{\bar{\rho}_1}\frac{d_1\bar{\rho}_1}{dt} - \frac{\alpha_2}{\bar{\rho}_2}\frac{d_2\bar{\rho}_2}{dt} \tag{3.23}$$

dove i termini a destra del segno di uguaglianza sono relativi alla comprimibilità dei fluidi, ovvero sono entrambi pari a 0 in caso di fluidi incomprimibili. Sostituendo all'interno del vincolo di continuità le espressioni 3.18 e 3.19 ottenute per i flussi, è possibile ricavare l'equazione della pressione:

$$\left[\nabla\cdot\left(\left(\alpha_{1,f}\left(\frac{1}{A_1\bar{\rho}_1}\right)_f + \alpha_{2,f}\left(\frac{1}{A_2\bar{\rho}_2}\right)_f\right)\nabla p\right)\right] = \nabla\cdot(\alpha_{1,f}\phi_1^* + \alpha_{2,f}\phi_2^*) + \\ + \frac{\alpha_1}{\bar{\rho}_1}\frac{d_1\bar{\rho}_1}{dt} + \frac{\alpha_2}{\bar{\rho}_2}\frac{d_2\bar{\rho}_2}{dt} \tag{3.24}$$

L'algoritmo di accoppiamento pressione e velocità può essere così riassunto:

1. risolvere le equazioni di trasporto della frazione volumica (eq. 3.11 e 3.9);
2. calcolare i coefficienti della forza di drag (eq. 3.13);
3. esplicitare le velocità nelle equazioni della quantità di moto nella forma delle eq. 3.16 e 3.17;
4. predire i flussi (eq. 3.20 e 3.21);
5. costruire e risolvere l'equazione della pressione (eq. 3.24);
6. correggere i flussi (eq. 3.18 e 3.19) con i valori di pressione ottenuti al punto precedente;
7. correggere le velocità (eq. 3.16 e 3.17) in base ai valori di pressione calcolati precedentemente;

8. verificare se si è raggiunta la convergenza, in caso contrario si avvia un processo iterativo che riparte dal punto 4, utilizzando gli ultimi valori di velocità calcolati per la predizione dei flussi.

L'algoritmo di accoppiamento pressione e velocità può essere schematizzato come in figura 3.1.

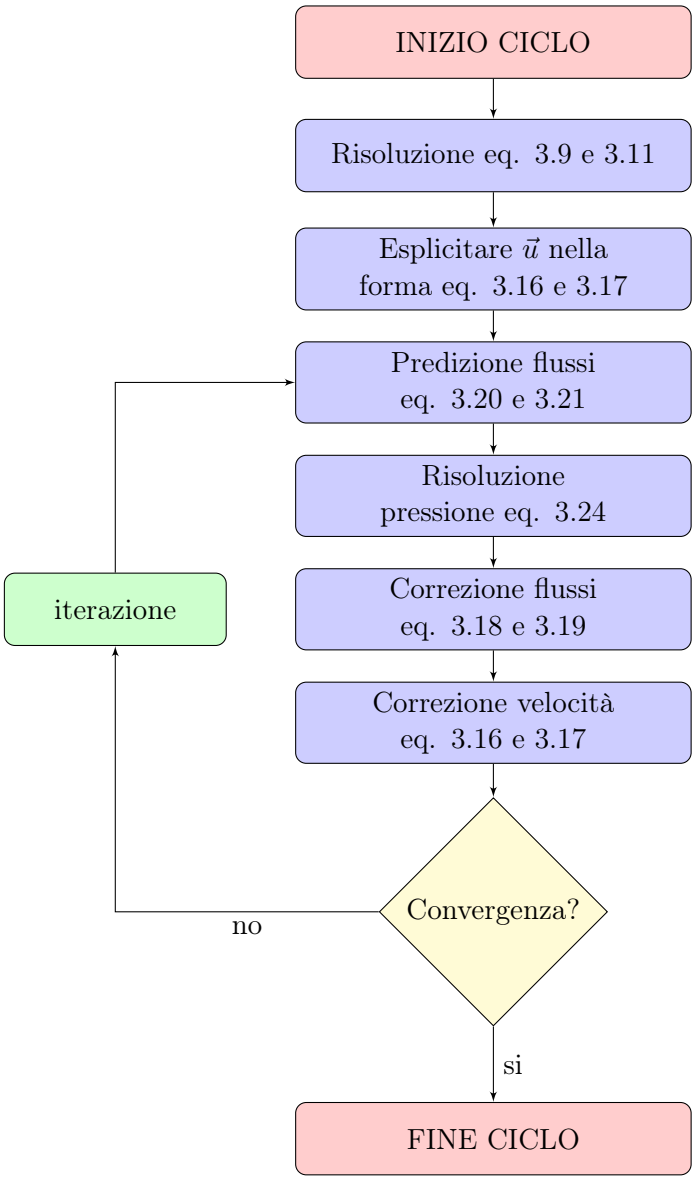

Figura 3.1: Diagramma di flusso dell'algoritmo iterativo di accoppiamento pressione e velocità per il codice `twoPhaseEulerFoam`.

Per la correzione dei flussi e delle velocità si utilizza l'algoritmo di eliminazione parziale (PEA), basato sul lavoro di Spalding [22], studiato e adottato da diversi autori [16], [13], [17] e [23].
Questo metodo consente un disaccoppiamento parziale di ciascuna equazione del bilancio di quantità di moto, che, in determinate condizioni, ha un impatto significativo sulla convergenza e stabilità della simulazione.

3.2 `MultiphaseEulerFoam`

Di seguito vengono descritte le equazioni che governano il modello Multi-fluid, riassunte nel paragrafo 2.4.2, nella forma implementata nel codice `multiphaseEulerFoam`.

3.2.1 Equazione di trasporto della frazione volumica

In `multiphaseEulerFoam` vengono considerati solamente fluidi incomprimibili, quindi le equazioni di trasporto della frazione volumica della fase continua (eq. 2.69) e delle fasi disperse (eq. 2.70) del modello Multi-fluid, sono state divise per la densità di ciascuna fase, considerata costante.
Per la fase continua si ottiene:

$$\frac{\partial \alpha_N}{\partial t} + \nabla \cdot (\alpha_N \hat{\vec{u}}_N) = 0 \tag{3.25}$$

Per le fasi disperse $k = 1, ..., N-1$:

$$\frac{\partial \alpha_k}{\partial t} + \nabla \cdot (\alpha_k \hat{\vec{u}}_k) = 0 \tag{3.26}$$

Le equazioni di trasporto della frazione volumica all'interno del codice vengono scritte nella forma riportata nelle equazioni 3.25 e 3.26, utilizzando un unico indice i per tutte le fasi presenti.
Tuttavia nelle equazioni di trasporto della frazione volumica compare un termine aggiuntivo che fa riferimento alla curvatura dell'interfaccia che solitamente si ha in simulazioni dettagliate di tracciamento dell'interfaccia e non in quelle euleriane-euleriane. Come è possibile notare dal codice sottostante, all'equazione della velocità relativa viene aggiunto un termine di curvatura dell'interfaccia denominato `nHatf`, funzione delle frazioni volumiche delle fasi disperse e di quella continua.

```
forAllIter(PtrDictionary<phaseModel>, phases_, iter2)
{
  phaseModel& phase2 = iter2();
  volScalarField& alpha2 = phase2;

  if (&phase2 == &phase) continue;

  surfaceScalarField phir(phase.phi() - phase2.phi());

  scalarCoeffSymmTable::const_iterator cAlpha
  (
    cAlphas_.find(interfacePair(phase, phase2))
  );

  if (cAlpha != cAlphas_.end())
   {
     surfaceScalarField phic
     (
      (mag(phi_) + mag(phir))/mesh_.magSf()
     );
```

```
        phir += min(cAlpha()*phic, max(phic))*nHatf(phase, phase2);
    }
}
```

Infine, una caratteristica del codice `multiphaseEulerFoam` è la definizione del campo `alphas`. Questo campo non rappresenta la frazione volumica di una determinata fase e pertanto non è limitato tra 0 e 1, ma viene utilizzato per rappresentare tutte le fasi in un singolo campo scalare.
`Alphas` viene calcolato sommando i prodotti dell'indice di fase e della frazione di fase.

$$\texttt{alphas} = \sum_{i=0}^{n-1} i * \alpha_i \tag{3.27}$$

Tutte le equazioni utilizzate nel codice sono riportate in appendice B.1.

3.2.2 Equazione di bilancio della quantità di moto

L'equazione di bilancio della quantità di moto per la fase primaria viene scritta come:

$$\frac{\partial \alpha_N \hat{\vec{u}}_N}{\partial t} + \nabla\cdot(\alpha_N \hat{\vec{u}}_N \hat{\vec{u}}_N) = \nabla\cdot(\alpha_N \nu_N(\nabla \hat{\vec{u}}_N + \nabla \hat{\vec{u}}_N^T)) - \nabla\cdot(\alpha_N \nu_N \frac{1}{3}\nabla \hat{\vec{u}}_N \overline{\overline{I}}) + \\ - \frac{\alpha_N}{\bar{\rho}_N}\nabla p + \alpha_N \vec{g} + \sum_{k=1}^{N-1} \frac{3}{4}\alpha_k \frac{C_{D,k}}{\bar{d}_{b,k}} |\hat{\vec{u}}_k - \hat{\vec{u}}_N|(\hat{\vec{u}}_k - \hat{\vec{u}}_N) \tag{3.28}$$

Per le fasi secondarie ($k = 1, ..., N-1$):

$$\frac{\partial \alpha_k \hat{\vec{u}}_k}{\partial t} + \nabla\cdot(\alpha_k \hat{\vec{u}}_k \hat{\vec{u}}_k) = \nabla\cdot(\alpha_k \nu_k(\nabla \hat{\vec{u}}_k + \nabla \hat{\vec{u}}_k^T)) - \nabla\cdot(\alpha_k \nu_k \frac{1}{3}\nabla \hat{\vec{u}}_k \overline{\overline{I}}) + \\ - \frac{\alpha_k}{\bar{\rho}_k}\nabla p + \alpha_k \vec{g} - \frac{3}{4}\frac{\bar{\rho}_N}{\bar{\rho}_k}\alpha_k \frac{C_{D,k}}{\bar{d}_{b,k}} |\hat{\vec{u}}_k - \hat{\vec{u}}_N|(\hat{\vec{u}}_k - \hat{\vec{u}}_N) \tag{3.29}$$

Nel codice le equazioni di bilancio per la quantità di moto vengono implementate come un set di equazioni, che viene definito in modo identico per tutte le fasi presenti:

```
UEqns.set
{
  phasei,
  new fvVectorMatrix
 (fvm::ddt(alpha, U)
  + fvm::div(phase.alphaPhi(), U)
  + (alpha/phase.rho())
      *fluid.Cvm(phase)
      *(fvm::ddt(U)
      + fvm::div(phase.phi(), U)
      - fvm::Sp(fvc::div(phase.phi()), U))
  - fvm::laplacian(alpha*nuEff, U)
  - fvc::div(alpha*(nuEff*dev(T(fvc::grad(U)))))
 ==
 (alpha/phase.rho())*fluid.Svm(phase)
```

```
  - fvm::Sp(slamDampCoeff
      *max(mag(U())
      - maxSlamVelocity,
        dimensionedScalar("U0", dimVelocity, 0))
        /pow(mesh.V(), 1.0/3.0),U))
}
```

dove il terzo termine è il contributo della forza di massa virtuale, che in questo caso è pari a 0, come anche il primo e il secondo termine a destra del segno di uguaglianza, che rappresentano rispettivamente il termine sorgente relativo alla forza di massa virtuale e quello relativo allo smorzamento che ridimensiona la velocità in base ad un valore massimo detto `maxSlamVelocity` e un coefficiente denominato `slamDampCoeff`.
Come si può notare dal codice, nell'equazione di bilancio della quantità di moto non è presente il termine di drag, che verrà aggiunto nel calcolo dell'equazione della pressione. Il termine di drag è definito, all'interno del codice, come segue:

```
Kptr =
(
   max(dm.phase1()*dm.phase2(),
       dm.residualPhaseFraction())
       *dm.K(max(mag(dm.phase1().U() - dm.phase2().U()),
        dm.residualSlip()))
).ptr();
```

dove il `residualPhaseFraction` è un valore che viene utilizzato nel caso in cui la frazione volumica tenda a zero, e il `residualSlip` rappresenta un valore che viene impiegato quando la velocità relativa tende a valori prossimi allo zero.
Si possono, quindi, riscrivere le linee di codice in forma di equazione, per una generica fase dispersa $k = 1, ..., N-1$:

$$K = \frac{3}{4}\alpha_k \alpha_N C_D \frac{\bar{\rho}_N}{\bar{d}_{b,k}} |\hat{\vec{u}}_r| \tag{3.30}$$

dove si ricorda che il pedice N indica la fase continua.

3.2.3 Algoritmo di accoppiamento pressione e velocità

Per la derivazione dell'equazione della pressione, si esplicitano le velocità espresse nel bilancio di quantità di moto come:

$$\vec{u}_k = \frac{H_k}{A_k} - \frac{\nabla p}{A_k} \tag{3.31}$$

dove A_k rappresenta la matrice dei coefficienti della velocità della fase k, e H_k è la matrice dei termini noti.
L'equazione per i flussi volumetrici alle facce può essere scritta come:

$$\phi_k = \phi_k^* - \left(\frac{1}{A_k \bar{\rho}_k}\right)_f S_f \nabla_f^{\perp} p \tag{3.32}$$

dove il termine fra parentesi corrisponde ad un'operazione di interpolazione alle facce usando il metodo di Rhie-Chow, e ϕ_k^* rappresenta la predizione del flusso per la fase k,

definito come:

$$\phi_k^* = \left(\frac{H_k}{A_k}\right)_f \cdot S_f \tag{3.33}$$

Il flusso volumetrico totale alle facce è:

$$\phi = \sum_{k=1}^{N} \alpha_{k,f}\phi_k \tag{3.34}$$

Utilizzando il vincolo di continuità $\nabla\cdot\phi = 0$, è possibile ricavare l'equazione della pressione:

$$\left[\nabla\cdot\left(\left(\sum_{k=1}^{N}\alpha_{k,f}\left(\frac{1}{A_k\bar{\rho}_k}\right)_f\right)\nabla p\right)\right] = \nabla\cdot\left(\sum_{k=1}^{N}\alpha_{k,f}\phi_k^*\right) \tag{3.35}$$

L'algoritmo di accoppiamento pressione e velocità può essere così riassunto:

1. risolvere le equazioni di trasporto della frazione volumica (eq. 3.25 e 3.26);
2. calcolare i coefficienti della forza di drag;
3. esplicitare le velocità nelle equazioni della quantità di moto nella forma dell'eq. 3.31;
4. predire i flussi (eq. 3.33);
5. costruire e risolvere l'equazione della pressione (eq. 3.35);
6. correggere i flussi (eq. 3.32) con i valori di pressione ottenuti al punto precedente;
7. correggere le velocità (eq. 3.31) in base ai valori di pressione calcolati precedentemente;
8. verificare se si è raggiunta la convergenza, in caso contrario si avvia un processo iterativo che riparte dal punto 4, utilizzando gli ultimi valori di velocità calcolati per la predizione dei flussi.

L'algoritmo di accoppiamento pressione e velocità può essere schematizzato come in figura 3.2.

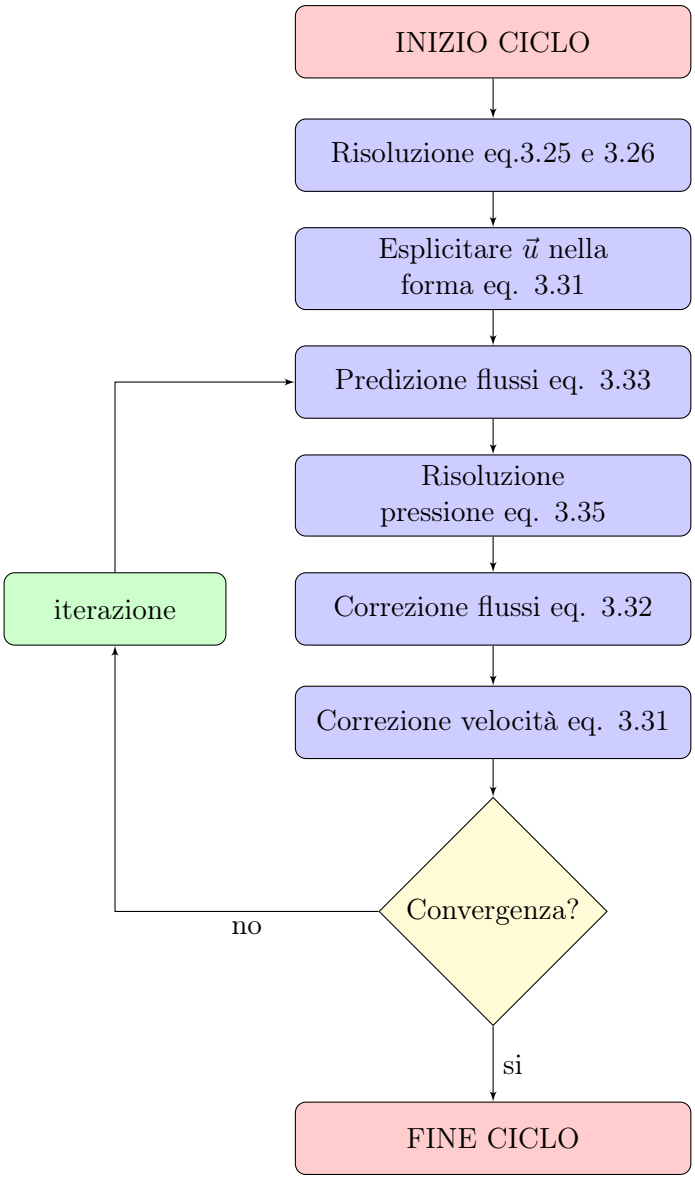

Figura 3.2: Diagramma di flusso dell'algoritmo iterativo di accoppiamento pressione velocità per il codice `multiphaseEulerFoam`.

3.3 Differenze tra i solver

Dopo aver parlato dell'implementazione di ciascun solver, è opportuno menzionare le differenze fra questi.
Una prima differenza, come si è potuto notare dai paragrafi 3.1.1 e 3.2.1, sta nell'implementazione dell'equazione di trasporto della frazione volumica della fase dispersa. In particolare:

- nel `twoPhaseEulerFoam` si considerano fluidi comprimibili e le equazioni vengono riscritte in funzione della velocità media e della velocità relativa per non avere problemi di instabilità numerica;

- nel `multiphaseEulerFoam` si considerano fluidi incomprimibili e le equazioni di trasporto della frazione volumica delle fasi disperse vengono scritte senza scomporre il termine convettivo.

Queste differenze possono essere visualizzate in colore rosso nei codici `twoPhaseEulerFoam` e `multiphaseEulerFoam`, rispettivamente indicati in appendice A.1 e B.1.
Una seconda differenza è rappresentata dalla scrittura delle equazioni di bilancio di quantità di moto, descritte nei paragrafi 3.1.2 e 3.2.2. Infatti:

- nel `twoPhaseEulerFoam` il termine di drag viene suddiviso in una parte esplicita ed una implicita, e solamente la parte implicita verrà considerata all'interno delle equazioni della quantità di moto, mentre la parte esplicita verrà inserita direttamente nell'equazione della pressione all'interno delle matrici dei termini noti, come già spiegato nel paragrafo 3.1.3;

- nel `multiphaseEulerFoam` la forza di drag per unità di volume viene aggiunta in un secondo momento, direttamente alle matrici costruite nel calcolo della pressione. La parte implicita viene aggiunta alle matrici dei coefficienti delle velocità, mentre la parte esplicita alle matrici dei termini noti.

In appendice A.2, A.5 e B.4 sono state evidenziate in verde le differenze sopra citate.
Per quanto riguarda il calcolo della forza di drag:

- nel `twoPhaseEulerFoam` viene utilizzata l'equazione 3.15, in cui si considera la frazione volumica della fase dispersa;

- nel `multiphaseEulerFoam` viene considerata anche la frazione volumica della fase continua, come si può vedere nell'equazione 3.30.

Questa differenza implica un andamento della forza di drag diverso fra i due codici, come mostrato in figura 3.3. L'andamento della forza di drag normalizzata all'aumentare della frazione volumica della fase dispersa va a 0 in `multiphaseEulerFoam`, mentre va a 1 in `twoPhaseEulerFoam`.
I confronti tra il `twoPhaseEulerFoam` e il `multiphaseEulerFoam` sono illustrati in blu rispettivamente nelle appendici A.3 e B.1.

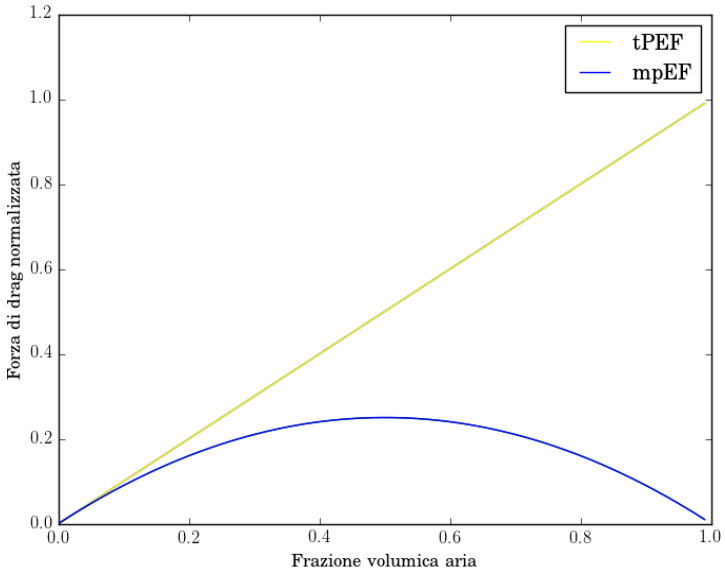

Figura 3.3: Andamento della forza di drag normalizzata in funzione della frazione volumica dell'aria, in `twoPhaseEulerFoam` e in `multiphaseEulerFoam`.

Un'altra differenza presente fra i due codici è relativa al termine `residualSlip`, di cui si è discusso nel paragrafo 3.2.2:

- nel `twoPhaseEulerFoam` il termine `residualSlip` non è definito e non viene utilizzato;
- nel `multiphaseEulerFoam`, viene utilizzato quando le velocità relative tendono a zero.

Questo termine è illustrato in viola in appendice B.1.
Analogamente avviene per il termine `residualRe`, definito nel paragrafo 3.1.2:

- nel `twoPhaseEulerFoam` viene utilizzato quando il numero di Reynolds tende a zero, per evitare una suddivisione per zero all'interno delle equazioni;
- nel `multiphaseEulerFoam`, non viene definito questo termine ma viene imposto un valore di default pari a 10^{-3}.

Queste differenze possono essere visualizzate in colore arancione nei codici indicati in appendice A.4 e B.3.
La sesta differenza fra i due codici è relativa al termine `residualAlpha`, descritto nel paragrafo 3.1.2:

- nel `twoPhaseEulerFoam` viene utilizzato quando la frazione volumica tende a zero, in aggiunta a tutte le matrici dei termini noti e dei coefficienti delle velocità, presenti nell'algoritmo di accoppiamento pressione e velocità;
- nel `multiphaseEulerFoam` questo termine viene denominato `residualPhaseFraction`, e non viene utilizzato nelle matrici dei termini noti e dei coefficienti delle velocità.

In appendice A.5 e B.4 vengono mostrate in marrone le differenze appena citate.
Infine, l'ultima differenza è dovuta al metodo di eliminazione parziale per la correzione dei flussi e delle velocità, di cui si è discusso nel paragrafo 3.1.3, che viene impiegato solamente nel codice `twoPhaseEulerFoam`.

3.4 Modifiche effettuate

3.4.1 Modifiche al codice `twoPhaseEulerFoam`

Le uniche modifiche effettuate nel codice `twoPhaseEulerFoam` sono relative al termine di drag, poiché, come si è visto nel paragrafo precedente, non è direttamente confrontabile con quello del `multiphaseEulerFoam`. Inizialmente si è optato per lasciare invariato il termine di drag del `twoPhaseEulerFoam` e modificare solamente quello del `multiphaseEulerFoam`, ma si sono riscontrati problemi numerici nelle simulazioni con il `multiphaseEulerFoam`.
Dunque, si è scelto di modificare il `twoPhaseEulerFoam` a partire dal coefficiente di drag, dividendolo per il valore di `Re`, come segue:

```
<Foam::volScalarField> CdRe() const
{
  volScalarField Re(max(pair_.Re(),residualRe_));

   return
    neg(Re - 1000)*24.0/Re*(1.0 + 0.15*pow(Re, 0.687))
    + pos0(Re - 1000)*0.44;
}
```

Poiché il termine `CdRe` è stato diviso per il numero di Reynolds, all'interno di `Ki`, ovvero il termine di forza di trascinamento per unità di volume diviso per la velocità relativa `Ur`, si è dovuto moltiplicare per il numero di Reynolds. Operando le opportune semplificazioni, si ottiene:

```
Foam::tmp<Foam::volScalarField> Foam::dragModel::Ki() const
{
  return
     0.75
  *CdRe()
  *swarmCorrection_->Cs()
  *pair_.continuous().rho()
  *pair_.magUr()
  /pair_.dispersed().d();
}
```

Infine un'ulteriore modifica effettuata per coerenza con l'altro solver è la moltiplicazione del termine `Ki` per il massimo tra le frazioni volumiche di entrambe le fasi e il `residualAlpha`.

```
Foam::tmp<Foam::volScalarField> Foam::dragModel::K() const
{
   return max(pair_.dispersed()* pair_.continuous(),
              pair_.dispersed().residualAlpha())*Ki();
}
```

L'equazione risulta ora equivalente a quella presente nel `multiphaseEulerFoam`:

$$K = \frac{3}{4}\alpha_1\alpha_2 C_D \frac{\bar{\rho}_2}{\bar{d}_{b,1}}|\hat{\vec{u}}_r| \tag{3.36}$$

Le modifiche effettuate nel codice sono mostrate in appendice C.1 e C.2.

3.4.2 Modifiche al codice `multiphaseEulerFoam`

La prima modifica effettuata nel `multiphaseEulerFoam` è stata l'introduzione del termine `residualAlpha`, laddove presente nel `twoPhaseEulerFoam`. In particolare è stato aggiunto alle matrici dei coefficienti delle velocità `A` e alle matrici dei termini noti `H`, come mostrato di seguito:

```
rAUs.set
(phasei, (1.0/(UEqns[phasei].A() + dragCoeffi
+ (max(phase.residualAlpha() - alpha, scalar(0))
 /runTime.deltaT()))).ptr());


HbyAs[phasei] = rAUs[phasei]*(UEqns[phasei].H()
+ max(phase.residualAlpha() - alpha, scalar(0))*U.oldTime()
/runTime.deltaT());
```

Ovviamente si sono modificate anche le restanti parti del codice dove si è utilizzata la matrice `A` o la matrice `H`. Per vedere tutte le modifiche effettuate, il lettore può fare riferimento in appendice D.4. Un'altra modifica effettuata è relativa all'interpolazione effettuata nel termine `rAlphaAUfs`, in modo del tutto analogo al `twoPhaseEulerFoam`:

```
rAlphaAUfs.set
 (
 phasei,
 (
 fvc::interpolate(max(alpha,phase.residualAlpha())*
  (1.0/(UEqns[phasei].A()
   + dragCoeffi + (max(phase.residualAlpha() - alpha, scalar(0))
    /runTime.deltaT()))))).ptr())
 );
```

Infine, l'ultima modifica è stata effettuata sul set di equazioni per la frazione volumica delle fasi disperse. Poiché, come visto in precedenza, l'implementazione del modello multifluido in `multiphaseEulerFoam` include termini aggiuntivi che fanno riferimento alla dinamica dell'interfaccia fra le fasi, si è deciso di riscrivere totalmente questa parte. Per semplificare questa operazione di scrittura del codice, si è deciso di scrivere 3 codici differenti, in cui le fasi disperse sono rispettivamente 1, 2 e 3. La generalizzazione per $N-1$ fasi disperse parte da queste.
Per il caso con una sola fase dispersa, l'equazione implementata nel codice è identica a quella ricavata nel paragrafo 3.1.1:

$$\frac{\partial \alpha_1}{\partial t} + \nabla\cdot(\alpha_1\hat{\vec{u}}) + \nabla\cdot(\alpha_1(1-\alpha_1)\hat{\vec{u}}_r) = 0 \tag{3.37}$$

Per la fase continua si utilizza la seguente equazione:

$$\alpha_2 = 1 - \alpha_1 \tag{3.38}$$

Il codice modificato per una fase dispersa è contenuto in Appendice D.1.
Nel caso di due fasi disperse, effettuando la stessa procedura di sostituzione del paragrafo 3.1.1, ovvero:

$$\hat{\vec{u}}_1 = \hat{\vec{u}} + (1 - \alpha_1)\hat{\vec{u}}_{r,1} - \alpha_2\hat{\vec{u}}_{r,2} \tag{3.39}$$

dove le velocità relative sono definite come:

$$\hat{\vec{u}}_{r,1} = \hat{\vec{u}}_1 - \hat{\vec{u}}_3 \tag{3.40}$$

$$\hat{\vec{u}}_{r,2} = \hat{\vec{u}}_2 - \hat{\vec{u}}_3 \tag{3.41}$$

È possibile ricavare le equazioni di trasporto della frazione volumica delle fasi disperse:

$$\frac{\partial \alpha_1}{\partial t} + \nabla\cdot(\alpha_1\hat{\vec{u}}) + \nabla\cdot(\alpha_1(1-\alpha_1)\hat{\vec{u}}_{r,1}) - \nabla\cdot(\alpha_1\alpha_2\hat{\vec{u}}_{r,2}) = 0 \tag{3.42}$$

$$\frac{\partial \alpha_2}{\partial t} + \nabla\cdot(\alpha_2\hat{\vec{u}}) + \nabla\cdot(\alpha_2(1-\alpha_2)\hat{\vec{u}}_{r,2}) - \nabla\cdot(\alpha_2\alpha_1\hat{\vec{u}}_{r,1}) = 0 \tag{3.43}$$

Per la fase continua:

$$\alpha_3 = 1 - \alpha_1 - \alpha_2 \tag{3.44}$$

Il codice modificato per 2 fasi disperse è mostrato in Appendice D.2.
Analogamente, per 3 fasi disperse:

$$\hat{\vec{u}}_1 = \hat{\vec{u}} + (1 - \alpha_1)\hat{\vec{u}}_{r,1} - \alpha_2\hat{\vec{u}}_{r,2} - \alpha_3\hat{\vec{u}}_{r,3} \tag{3.45}$$

dove le velocità relative sono definite come:

$$\hat{\vec{u}}_{r,1} = \hat{\vec{u}}_1 - \hat{\vec{u}}_4 \tag{3.46}$$

$$\hat{\vec{u}}_{r,2} = \hat{\vec{u}}_2 - \hat{\vec{u}}_4 \tag{3.47}$$

$$\hat{\vec{u}}_{r,3} = \hat{\vec{u}}_3 - \hat{\vec{u}}_4 \tag{3.48}$$

È possibile definire le equazioni di trasporto della frazione volumica delle fasi disperse:

$$\frac{\partial \alpha_1}{\partial t} + \nabla\cdot(\alpha_1\hat{\vec{u}}) + \nabla\cdot(\alpha_1(1-\alpha_1)\hat{\vec{u}}_{r,1}) - \nabla\cdot(\alpha_1\alpha_2\hat{\vec{u}}_{r,2}) - \nabla\cdot(\alpha_1\alpha_3\hat{\vec{u}}_{r,3}) = 0 \tag{3.49}$$

$$\frac{\partial \alpha_2}{\partial t} + \nabla\cdot(\alpha_2\hat{\vec{u}}) + \nabla\cdot(\alpha_2(1-\alpha_2)\hat{\vec{u}}_{r,2}) - \nabla\cdot(\alpha_2\alpha_1\hat{\vec{u}}_{r,1}) - \nabla\cdot(\alpha_2\alpha_3\hat{\vec{u}}_{r,3}) = 0 \tag{3.50}$$

$$\frac{\partial \alpha_3}{\partial t} + \nabla\cdot(\alpha_3\hat{\vec{u}}) + \nabla\cdot(\alpha_3(1-\alpha_3)\hat{\vec{u}}_{r,3}) - \nabla\cdot(\alpha_3\alpha_1\hat{\vec{u}}_{r,1}) - \nabla\cdot(\alpha_3\alpha_2\hat{\vec{u}}_{r,2}) = 0 \tag{3.51}$$

Per la fase continua:

$$\alpha_4 = 1 - \alpha_1 - \alpha_2 - \alpha_3 \tag{3.52}$$

Il codice modificato per 3 fasi disperse è inserito in Appendice D.3.
Più in generale è possibile scrivere la seguente relazione per una generica fase i dispersa:

$$\hat{\vec{u}}_i = \hat{\vec{u}} + (1 - \alpha_i)\hat{\vec{u}}_{r,i} - \sum_{j\neq i}\alpha_j\hat{\vec{u}}_{r,j} \tag{3.53}$$

dove j indica una generica fase dispersa diversa da i.
Le velocità relative rispettivamente per le fasi disperse i e j vengono definite come:

$$\hat{\vec{u}}_{r,i} = \hat{\vec{u}}_i - \hat{\vec{u}}_N \tag{3.54}$$

$$\hat{\vec{u}}_{r,j} = \hat{\vec{u}}_j - \hat{\vec{u}}_N \tag{3.55}$$

in cui la fase N rappresenta quella continua.
L'equazione di trasporto della frazione volumica di una generica fase dispersa i, può essere quindi scritta come:

$$\frac{\partial \alpha_i}{\partial t} + \nabla\cdot(\alpha_i \hat{\vec{u}}) + \nabla\cdot(\alpha_i(1-\alpha_i)\hat{\vec{u}}_{r,i}) - \alpha_i \sum_{j\neq i} \alpha_j \hat{\vec{u}}_{r,j} = 0 \tag{3.56}$$

Per la fase continua:

$$\alpha_N = 1 - \sum_{i=1}^{N-1} \alpha_i \tag{3.57}$$

dove l'indice i include tutte le $N-1$ fasi disperse considerate.

Capitolo 4

Descrizione test case

4.1 Struttura generale di una simulazione in OpenFOAM

La risoluzione di un problema fluidodinamico con il codice CFD OpenFOAM avviene attraverso la creazione di una cartella chiamata *case*, all'interno del quale sono contenute tutte le informazioni riguardanti la griglia, le condizioni iniziali e al contorno, i parametri necessari per il tipo di simulazione che si vuole effettuare, e infine i risultati ottenuti mediante il solver utilizzato. Queste informazioni sono collocate in diverse sottocartelle, come mostrato in figura 4.1.
La prima sottocartella è "system", nella quale sono presenti tre file di testo:

- "controlDict": comprende alcuni importanti parametri della simulazione, tra cui i parametri dell'integrazione temporale, come il tempo di inizio e di fine integrazione e il numero di Courant massimo, per il calcolo del passo di integrazione temporale adattativo. Nella sezione "function object" sono presenti tutte le specifiche delle grandezze che si vogliono calcolare in fase di post processing dei risultati;

- "fvSchemes": contiene indicazioni sugli schemi di discretizzazione nello spazio e nel tempo utili allo svolgimento della simulazione;

- "fvSolution": include la definizione dei solutori numerici utilizzati per risolvere le equazioni discretizzate, le tolleranze di iterazione del calcolo, i coefficienti di sottorilassamento e i parametri relativi all'algoritmo di accoppiamento pressione e velocità.

La seconda è la cartella "constant", che contiene tutti i file inerenti la definizione dei modelli utilizzati nella simulazione, come "transportProperties" e "turbulenceProperties" che definiscono i valori di alcune grandezze fisiche dei fluidi come la viscosità e la densità, e il modello di turbolenza utilizzato. La sottocartella "polyMesh" viene originata solo dopo aver creato la griglia e include una serie di file testuali che descrivono tutte le informazioni necessarie al codice per ricostruire la topologia della griglia di calcolo.
La terza cartella è in realtà un gruppo di cartelle chiamato "time directories", che comprende la prima cartella necessaria all'avvio della simulazione chiamata "0" in cui vengono specificate le condizioni iniziali e al contorno relative alle grandezze incognite. Una volta avviata la simulazione, vengono generate nuove cartelle contenenti i risultati del calcolo, dove il nome di ogni directory temporale è dato dal tempo della simulazione nel quale il dato viene scritto. La frequenza di questi salvataggi si può modificare nel file "controlDict".

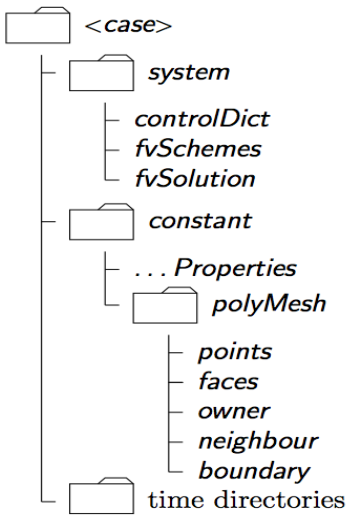

Figura 4.1: Rappresentazione della struttura di un caso in OpenFOAM.

4.2 Caso in esame

Come menzionato nei precedenti capitoli, l'obiettivo del lavoro è verificare l'utilizzo del codice `multiphaseEulerFoam` per simulare sistemi multifase con l'approccio multifluido. Per questo si è scelto di confrontare il `multiphaseEulerFoam` con il codice `twoPhaseEulerFoam`, già utilizzato in precedenza [17] per la simulazione di sistemi multifase.
Per questo motivo, in questo lavoro, sono stati definiti diversi test case, in cui sono state specificate le stesse condizioni iniziali e al contorno, le stesse proprietà fisicochimiche, gli stessi parametri temporali e gli stessi metodi numerici per le simulazioni eseguite con entrambi i codici, in modo tale da operare un confronto coerente tra il `twoPhaseEulerFoam` e il `multiphaseEulerFoam`. In particolare per analizzare la capacità del `multiphaseEulerFoam` nel simulare più fasi, sono state eseguite 3 simulazioni differenti, in cui la stessa fase dispersa è stata ripartita rispettivamente in una, due o tre fasi identiche.
È possibile così riassumere le 4 simulazioni confrontate:

- `twoPhaseEulerFoam`: 1 fase continua e 1 fase dispersa;
- `multiphaseEulerFoam`: 1 fase continua e 1 fase dispersa;
- `multiphaseEulerFoam`: 1 fase continua e 2 fasi disperse;
- `multiphaseEulerFoam`: 1 fase continua e 3 fasi disperse.

4.2.1 Geometria e condizioni operative

Il sistema multifase oggetto di studio è quello studiato sperimentalmente e numericamente da Diaz et al. [9] e da Buffo et al. [5], ed è composto da una colonna a bolle con sezione rettangolare di altezza 0.66 m, larghezza 0.2 m e profondità 0.04 m, come mostrato in figura 4.2.
L'acqua costituisce la fase continua nella quale sono disperse le bolle d'aria, che gorgogliano all'interno della colonna per mezzo di uno sparger. Quest'ultimo si trova esattamente

al centro della base della colonna ed è costituito da 8 fori di 1 mm distanziati fra loro con un passo di 6 mm. Poichè la simulazione accurata della formazione delle bolle allo sparger richiederebbe un livello di dettaglio troppo oneroso dal punto di vista computazionale, si è scelto di rappresentare lo sparger con una superficie di lunghezza di 0.024 m e larghezza di 0.012 m, in cui la portata di gas sotto forma di bolle entra uniformemente. La colonna è riempita con dell'acqua sino ad un livello di 0.45 m.

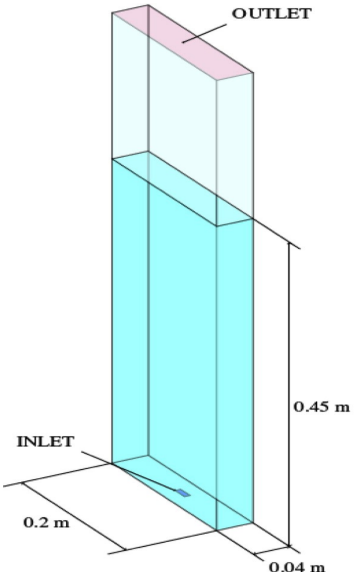

Figura 4.2: Rappresentazione schematica della colonna a bolle simulata [5].

Per la creazione della mesh si è utilizzato un file.m4 che ha permesso di generare un file "blockMeshDict". In questo file è specificata la suddivisione della geometria della colonna in parete anteriore (side wall) e posteriore (bottom wall), e in sezione di ingresso (inlet) e di uscita (outlet), oltre che i dettagi relativi alla griglia computazionale.
In questo lavoro è stata utilizzata una tipologia di griglia con celle esaedriche, composta da 32 celle lungo la larghezza della colonna, 11 lungo la profondità e 70 lungo l'altezza, per un totale di 24640 celle; per lo sparger, invece, si sono utilizzate 6 celle lungo la larghezza e 3 lungo la profondità. Questa griglia è la stessa utilizzata nel lavoro di Buffo et al. [5] e verrà chiamata con l'etichetta utilizzata in precedenza, ovvero "griglia media".

4.2.2 Condizioni iniziali e al contorno

Per la temperatura della fase continua, le condizioni iniziali del campo sono state imposte uniformi e pari alla temperatura in ingresso di 300 K. La condizione *fixedValue* impone un valore costante di temperatura pari a 300 K. Analogamente, per la fase dispersa il valore iniziale di temperatura è pari a 300 K. La condizione *zeroGradient* implica che i gradienti di temperatura siano nulli alle pareti. Tutto ciò ha permesso di trascurare gli effetti dovuti allo scambio di calore fra le fasi e con le pareti (sistema isotermo).

Pressione	**twoPhase EulerFoam**	**multiphase EulerFoam**		
	1 fase	1 fase	2 fasi	3 fasi
Inlet				
type	*fixedFlux-Pressure*	*fixedFlux-Pressure*	*fixedFlux-Pressure*	*fixedFlux-Pressure*
value	uniform 10^5	uniform 10^5	uniform 10^5	uniform 10^5
Outlet				
type	*prghPressure*	*prghPressure*	*prghPressure*	*prghPressure*
p	uniform 10^5	uniform 10^5	uniform 10^5	uniform 10^5
value	uniform 10^5	uniform 10^5	uniform 10^5	uniform 10^5
Side wall				
type	*fixedFlux-Pressure*	*fixedFlux-Pressure*	*fixedFlux-Pressure*	*fixedFlux-Pressure*
value	uniform 10^5	uniform 10^5	uniform 10^5	uniform 10^5
Bottom wall				
type	*fixedFlux-Pressure*	*fixedFlux-Pressure*	*fixedFlux-Pressure*	*fixedFlux-Pressure*

Tabella 4.1: Condizioni al contorno della pressione per le simulazioni a seconda del numero di fasi disperse.

Per quanto concerne la pressione, quella a cui si fa riferimento è la pressione diminuita del termine idrostatico, ed è definita come:

$$p_{rgh} = p - \rho g h \tag{4.1}$$

dove p è la pressione statica e h l'altezza idrostatica.
La condizione iniziale dell'*internal field* è settata pari a $10^5 Pa$. In tabella 4.1 vengono indicate le condizioni al contorno per la pressione.
La condizione *fixedFluxPressure* viene applicata quando sono presenti forze come la gravità e la tensione superficiale, e permette di regolarne il gradiente.

Frazione volumica fase continua	**twoPhase EulerFoam**	**multiphase EulerFoam**		
	1 fase	1 fase	2 fasi	3 fasi
Inlet				
type	*fixedValue*	*fixedValue*	*fixedValue*	*fixedValue*
value	uniform 0.5	uniform 0.5	uniform 0.5	uniform 0.5
Outlet				
type	*inletOutlet*	*inletOutlet*	*inletOutlet*	*inletOutlet*
phi	phi.air	phi.air	phi.air	phi.air
inletValue	uniform 0	uniform 0	uniform 0	uniform 0
value	uniform 0	uniform 0	uniform 0	uniform 0
Side wall				
type	*zeroGradient*	*zeroGradient*	*zeroGradient*	*zeroGradient*
Bottom wall				
type	*zeroGradient*	*zeroGradient*	*zeroGradient*	*zeroGradient*

Tabella 4.2: Condizioni al contorno della frazione volumica della fase continua per le simulazioni a seconda del numero di fasi disperse.

Frazione volumica fase dispersa	**twoPhase EulerFoam** 1 fase	**multiphase EulerFoam** 1 fase	2 fasi	3 fasi
Inlet				
type	*fixedValue*	*fixedValue*	*fixedValue*	*fixedValue*
value	uniform 0.5	uniform 0.5	uniform 0.25	uniform 0.166
Outlet				
type	*inletOutlet*	*inletOutlet*	*inletOutlet*	*inletOutlet*
phi	phi.air	phi.air	phi.air	phi.air
inletValue	uniform 1	uniform 1	uniform 1	uniform 1
value	uniform 1	uniform 1	uniform 1	uniform 1
Side wall				
type	*zeroGradient*	*zeroGradient*	*zeroGradient*	*zeroGradient*
Bottom wall				
type	*zeroGradient*	*zeroGradient*	*zeroGradient*	*zeroGradient*

Tabella 4.3: Condizioni al contorno della frazione volumica della fase dispersa per le simulazioni a seconda del numero di fasi disperse.

Per le frazione volumiche, come condizioni iniziali, si sono imposti i valori di campo in una regione specifica, che permette di definire l'insieme di celle che delimitano il livello d'acqua. La frazione volumica di acqua è 1 in questa regione e 0 nella parte restante, mentre per le fasi disperse avviene l'opposto.
Nelle tabelle 4.2 e 4.3 vengono indicate rispettivamente le condizioni al contorno per la frazione volumica della fase continua e delle fasi disperse. Il valore della frazione volumica delle fasi disperse all'*inlet* dipende dal numero di fasi secondarie presenti, ed è scelto in modo tale che la somma di tutte le frazioni volumiche delle fasi disperse sia pari a 0.5.
La condizione all'uscita è di tipo *inletOutlet*, ed è uguale alla condizione *zeroGradient* quando i flussi sono uscenti, mentre se i flussi sono entranti dalla frontiera di uscita, viene assegnato il valore della variabile in ingresso.
Il campo di velocità iniziale è stato impostato in modo da partire dalla condizione di fluido fermo in ogni parte del dominio dove è presente l'acqua. Le condizioni al contorno per la velocità della fase continua sono indicate in tabella 4.4.
La condizione *pressureInletOutletVelocity* permette di definire la velocità in ingresso a partire dalle condizioni di pressione note, e anche in questo caso è equivalente alla condizione di *zeroGradient* per flussi uscenti, e a quella di *fixedValue* pari al valore della variabile in ingresso per flussi entranti.
La velocità del flusso d'aria in ingresso viene calcolata tramite la seguente relazione:

$$\vec{u}_k^{IN} = \frac{Q}{\sum_{k=1}^{N-1} \alpha_k A_s} \tag{4.2}$$

dove Q rappresenta la portata d'aria che fluisce all'interno della colonna ed è settata pari a $1.92 \times 10^{-5}\ m^3/s$, A_s è l'area dello sparger pari a $2.88 \times 10^{-4}\ m^2$ e la sommatoria rappresenta la frazione volumica complessiva delle fasi disperse. Le condizioni al contorno per la velocità delle fasi disperse sono riassunte in tabella 4.5. Alle pareti viene imposta la condizione di slip, che permette al codice di trascurare gli effetti viscosi sul bordo del dominio su cui è applicata.

Velocità fase continua	**twoPhase EulerFoam** 1 fase	**multiphase EulerFoam** 1 fase	2 fasi	3 fasi
Inlet				
type	*fixedValue*	*fixedValue*	*fixedValue*	*fixedValue*
value	uniform (0 0 0)	uniform (0 0 0)	uniform (0 0 0)	uniform (0 0 0)
Outlet				
type	*pressureInlet-OutletVelocity*	*pressureInlet-OutletVelocity*	*pressureInlet-OutletVelocity*	*pressureInlet-OutletVelocity*
phi	phi.water	phi.water	phi.water	phi.water
value	uniform (0 0 0)	uniform (0 0 0)	uniform (0 0 0)	uniform (0 0 0)
Side wall				
type	*fixedValue*	*fixedValue*	*fixedValue*	*fixedValue*
value	uniform (0 0 0)	uniform (0 0 0)	uniform (0 0 0)	uniform (0 0 0)
Bottom wall				
type	*fixedValue*	*fixedValue*	*fixedValue*	*fixedValue*
value	uniform (0 0 0)	uniform (0 0 0)	uniform (0 0 0)	uniform (0 0 0)

Tabella 4.4: Condizioni al contorno della velocità della fase continua per le simulazioni a seconda del numero di fasi disperse.

Velocità fase dispersa	**twoPhase EulerFoam** 1 fase	**multiphase EulerFoam** 1 fase	2 fasi	3 fasi
Inlet				
type	*fixedValue*	*fixedValue*	*fixedValue*	*fixedValue*
value	uniform (0 0 0.133)	uniform (0 0 0.133)	uniform (0 0 0.133)	uniform (0 0 0.133)
Outlet				
type	*pressureInlet-OutletVelocity*	*pressureInlet-OutletVelocity*	*pressureInlet-OutletVelocity*	*pressureInlet-OutletVelocity*
phi	phi.air	phi.air	phi.air	phi.air
value	uniform (0 0 10^{-6})	uniform (0 0 10^{-6})	uniform (0 0 10^{-6})	uniform (0 0 10^{-6})
Side wall				
type	*slip*	*slip*	*slip*	*slip*
Bottom wall				
type	*slip*	*slip*	*slip*	*slip*

Tabella 4.5: Condizioni al contorno della velocità della fase dispersa per le simulazioni a seconda del numero di fasi disperse.

4.2.3 Proprietà delle fasi

In tabella 4.6 vengono specificati i valori relativi alle proprietà delle fasi. Sono stati utilizzati gli stessi valori di densità e viscosità cinematica per le fasi disperse di tutte le simulazioni, analogamente è stato fatto per la fase continua.
Il diametro delle bolle per tutte le fasi disperse è stato definito inizialmente di un 1 mm, e successivamente sono state effettuate delle simulazioni con valori di 10 mm.
Per quanto riguarda le forze di interazione fra le fasi, l'unica forza considerata è stata quella di drag, e il modello utilizzato è stato quello di Schiller-Naumann per tutte le simulazioni, trattando le bolle come sfere rigide. Il `residualRe` è stato impostato pari a

10^{-3} nel test case per il `twoPhaseEulerFoam`, per essere coerente con il valore prefissato di 10^{-3} definito all'interno del `multiphaseEulerFoam`, come spiegato nel paragrafo 3.3.
Il `residualAlpha` dell'acqua è pari a 10^{-3} per tutte le simulazioni, mentre per l'aria è stato scelto in modo tale che la somma dei `residualAlpha` di tutte le fasi disperse sia pari a 2×10^{-3}.
Infine è importante sottolineare che in tutte le simulazioni non è stato adottato alcun modello di turbolenza, per semplificare il confronto tra i codici.

Proprietà	**twoPhase EulerFoam**	**multiphase EulerFoam**		
	1 fase	1 fase	2 fasi	3 fasi
ρ_N $[kg/m^3]$	998.2	998.2	998.2	998.2
ν_N $[m^2/s]$	10^{-6}	10^{-6}	10^{-6}	10^{-6}
per $k = 1, ..., N-1$ ρ_k $[kg/m^3]$	$k = 1$ 1	$k = 1$ 1	$k = 1, 2$ 1	$k = 1, 2, 3$ 1
per $k = 1, ..., N-1$ ν_k $[m^2/s]$	$k = 1$ 1.6×10^{-5}	$k = 1$ 1.6×10^{-5}	$k = 1, 2$ 1.6×10^{-5}	$k = 1, 2, 3$ 1.6×10^{-5}
per $k = 1, ..., N-1$ d_k $[mm]$	$k = 1$ 1	$k = 1$ 1	$k = 1, 2$ 1	$k = 1, 2, 3$ 1
`residualRe`	10^{-3}	10^{-3}	10^{-3}	10^{-3}
`residualAlpha`(N)	10^{-3}	10^{-3}	10^{-3}	10^{-3}
per $k = 1, ..., N-1$ `residualAlpha`(k)	$k = 1$ 2×10^{-3}	$k = 1$ 2×10^{-3}	$k = 1, 2$ 10^{-3}	$k = 1, 2, 3$ 0.66×10^{-3}

Tabella 4.6: Proprietà delle fasi per le simulazioni a seconda del numero di fasi disperse.

4.2.4 Schemi numerici

Gli schemi numerici adottati per la risoluzione delle equazioni vengono indicati in tabella 4.6.
Per la discretizzazione delle derivate temporali ($\frac{\partial \Psi}{\partial t}$) si è scelto uno schema di Eulero implicito del primo ordine.
Per quanto riguarda i gradienti ($\nabla \Psi$) si è utilizzato uno schema di Gauss, con interpolazione tra i centri delle celle e i centri delle facce di tipo lineare.
Lo stesso schema viene utilizzato per i laplaciani ($\nabla^2 \Psi$), per i quali non si è adottato un metodo di correzione dell'errore introdotto dalla non-ortogonalità della mesh sui termini diffusivi, poiché la griglia computazionale adottata è cartesiana .
Per le interpolazioni ($\Psi \mid_f$) si è optato per un metodo lineare, mentre per i gradienti normali alla superficie ($\nabla_{\perp A} \Psi$) uno schema senza correzione della non-ortogonalità, per lo stesso motivo dello schema adottato per i termini laplaciani.
Infine vengono specificati gli schemi utilizzati per la discretizzazione dei termini convettivi:

- Gauss con schema di secondo ordine TVD (Total Variation Diminishing) e limitato secondo Van Leer per i termini di convezione della frazione volumica ($\nabla \cdot \alpha_i \vec{u}_i$);
- Gauss con schema TVD lineare e limitato per i termini di convezione della quantità di moto ($\nabla \cdot \alpha_i \vec{u}_i \vec{u}_i$);
- Gauss lineare per le componenti deviatoriche del tensore τ ($\nabla \cdot \tau_{dev}$).

Schemi numerici	**twoPhase EulerFoam**	**multiphase EulerFoam**		
	1 fase	1 fase	2 fasi	3 fasi
$\frac{\partial \Psi}{\partial t}$	Eulero	Eulero	Eulero	Eulero
$\nabla \Psi$	Gauss lineare	Gauss lineare	Gauss lineare	Gauss lineare
$\nabla^2 \Psi$	Gauss lineare non corretto	Gauss lineare non corretto	Gauss lineare non corretto	Gauss lineare non corretto
$\Psi \mid_f$	lineare	lineare	lineare	lineare
$\nabla_{\perp A} \Psi$	non corretto	non corretto	non corretto	non corretto
$\nabla \cdot \alpha_i \vec{u}_i$	Gauss vanLeer	Gauss vanLeer	Gauss vanLeer	Gauss vanLeer
$\nabla \cdot \alpha_i \vec{u}_i \vec{u}_i$	Gauss lineare limitato	Gauss lineare limitato	Gauss lineare limitato	Gauss lineare limitato
$\nabla \cdot \tau_{dev}$	Gauss lineare	Gauss lineare	Gauss lineare	Gauss lineare

Tabella 4.7: Schemi numerici per le simulazioni a seconda del numero di fasi disperse.

4.2.5 Schemi numerici per le equazioni discretizzate

In tabella 4.2 vengono indicati i tipi di solutori numerici per le equazioni discretizzate e le tolleranze utilizzate per la risoluzione delle equazioni di velocità e pressione, e il numero di cicli e correttori utilizzati per la risoluzione delle equazioni della frazione volumica e dell'algoritmo PIMPLE.
Quest'ultimo accorpa gli algoritmi PISO e SIMPLE, ed è stato creato per coniugare i benefici di entrambi gli approcci: il sotto rilassamento derivante dal SIMPLE assicura una rapida convergenza anche per time step relativamente piccoli, mentre un'iterazione finale senza rilassamento (proveniente dal PISO) assicura la consistenza temporale alla soluzione [4].

Schemi numerici equazioni	**twoPhase EulerFoam** 1 fase	**multiphase EulerFoam** 1 fase	2 fasi	3 fasi
Velocità				
solver	smoothSolver	smoothSolver	smoothSolver	smoothSolver
smoother	symGauss Seidel	symGauss Seidel	symGauss Seidel	symGauss Seidel
tolleranza	10^{-5}	10^{-5}	10^{-5}	10^{-5}
Pressione				
solver	GAMG	GAMG	GAMG	GAMG
smoother	DIC	DIC	DIC	DIC
tolleranza	10^{-8}	10^{-8}	10^{-8}	10^{-8}
Frazione volumica aria				
n° correttori	3	3	3	3
n° sottocicli	3	3	3	3
PIMPLE				
n° correttori in uscita	10	10	10	10
n° correttori	3	3	3	3
n° correttori non ortogonali	0	0	0	0
fattori di rilassamento	1	1	1	1

Tabella 4.8: Schemi numerici per le equazioni discretizzate per le simulazioni a seconda del numero di fasi disperse.

4.2.6 Integrazione temporale

Le ultime specifiche necessarie per inizializzare la simulazione sono quelle relative ai parametri per l'integrazione temporale. Inizialmente è necessario impostare il tempo di avvio (t=0 s), così che OpenFOAM possa leggere i dati dalla cartella "0", e il tempo di arresto della simulazione pari a 100 s.
L'intervallo temporale utilizzato è stato scelto in modo tale da rispettare la condizione di CFL (Courant–Friedrichs–Lewy):

$$CFL = \Delta t \sum_{i=1}^{3} \frac{\vec{u}_{r,i}}{\Delta x_i} < 1 \tag{4.3}$$

dove Δt rappresenta il time step, Δx_i è la spaziatura della griglia di calcolo lungo la direzione i e $\vec{u}_{r,i}$ è la velocità relativa tra il gas e il liquido lungo la direzione i.
Per ottenere precisione temporale e stabilità numerica si è utilizzato un time step massimo di $10^{-3}s$ e un numero di Courant massimo di 0.5. Sono state effettuate simulazioni sia con un time step fisso di $10^{-3}s$ sia con un time step variabile.
Per lo svolgimento delle simulazioni è stato impiegato un Cluster del Politecnico di Torino fornito da HPC@POLITO (http://www.hpc.polito.it).

Capitolo 5

Risultati

5.1 Risultati preliminari

Nella fase iniziale di questo lavoro, si sono confrontati i codici `twoPhaseEulerFoam` e `multiphaseEulerFoam`, presenti nella versione 5.0 di OpenFOAM, per verificare il modello implementato nel codice `multiphaseEulerFoam`.
Per fare ciò, si sono comparati i profili, mediati nel tempo (escludendo il transitorio iniziale), delle frazioni volumiche dell'aria, delle velocità dell'aria e dell'acqua, in funzione della distanza lungo l'asse x normalizzata rispetto alla larghezza della colonna (d), ad un'altezza pari alla metà del livello di liquido.
In figura 5.1 sono illustrati i profili delle frazioni volumiche di aria mediati nel tempo in funzione della lunghezza normalizzata, in giallo viene indicato il profilo ottenuto per il codice `twoPhaseEulerFoam` (tPEF), in blu quello per il `multiphaseEulerFoam` (mpEF) con 1 fase dispersa, in rosso quello con 2 fasi disperse e in verde quello con 3 fasi disperse. Nei casi in cui sono presenti più fasi secondarie, si è diagrammata la somma delle frazioni volumiche di tutte le fasi disperse. Dal grafico è possibile notare come il profilo della frazione volumica dell'aria in funzione della lunghezza normalizzata, ottenuto con il codice `twoPhaseEulerFoam`, sia nettamente differente da quelli conseguiti con il `multiphaseEulerFoam`.
Con il primo solver si ottiene un profilo più ampio ma con valori della frazione volumica nettamente inferiori, infatti al centro della colonna il valore massimo è di circa 0.04, mentre con il `multiphaseEulerFoam` si hanno dei profili maggiormente concentrati al centro della colonna con valori di frazione volumica locale fino a 4 volte più grandi rispetto alla soluzione ottenuta con il modello two-fluid.
Per quanto riguarda le simulazioni con il `multiphaseEulerFoam`, invece, si riscontrano profili quasi del tutto coincidenti, poiché sono state utilizzate delle impostazioni equivalenti per tutte le simulazioni, sia per quanto riguarda le condizioni iniziali e al contorno, che per quanto riguarda i parametri numerici del risolutore.

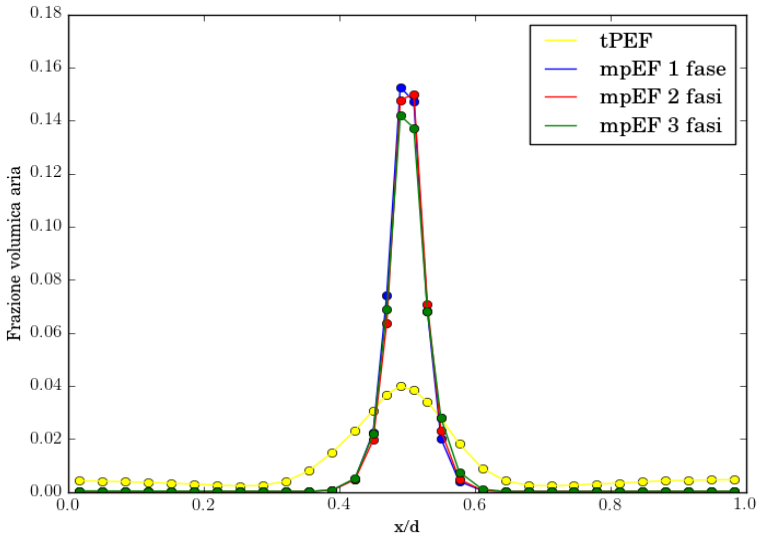

Figura 5.1: Profili delle frazioni volumiche dell'aria, mediati nel tempo, in funzione della distanza lungo l'asse x normalizzata rispetto alla larghezza della colonna, per le simulazioni con il `twoPhaseEulerFoam` e il `multiphaseEulerFoam`.

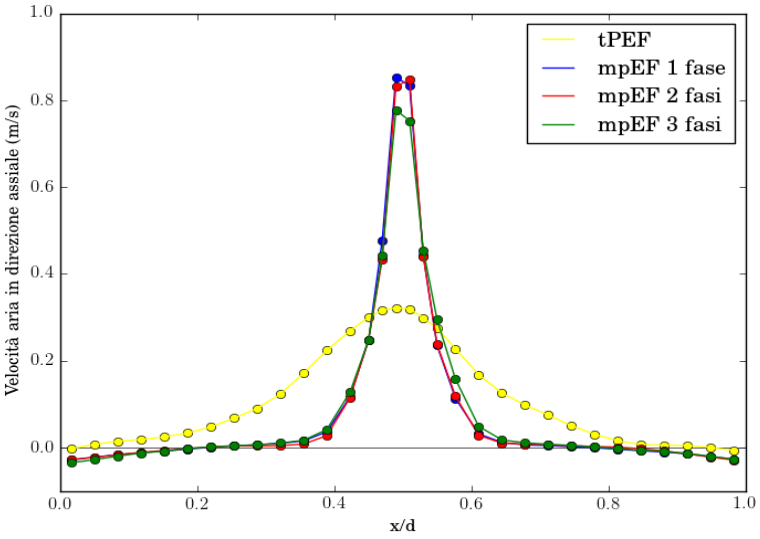

Figura 5.2: Profili di velocità assiale dell'aria, mediati nel tempo, in funzione della distanza lungo l'asse x normalizzata rispetto alla larghezza della colonna, per le simulazioni con il `twoPhaseEulerFoam` e il `multiphaseEulerFoam`.

I profili di velocità dell'aria in funzione della distanza lungo l'asse x normalizzata rispetto alla larghezza della colonna, sono illustrati in figura 5.2. Nei casi in cui sono presenti più fasi disperse, si è diagrammata una sola componente di velocità in direzione assiale per ogni fase dispersa, poiché le velocità risultano identiche per tutte le fasi disperse, come ci si aspetta. Infatti le diverse fasi disperse sono composte da bolle delle stesse dimensioni, e poiché la distribuzione dimensionale è identica per tutte le fasi disperse, le bolle si muovono con le stesse velocità.
Per quanto riguarda i profili di velocità dell'aria in funzione della lunghezza normalizzata,

è possibile constatare, come nel caso precedente, che il profilo di velocità ottenuto con il `twoPhaseEulerFoam` è nettamente differente da quelli ricavati con il `multiphaseEulerFoam`. Con il `twoPhaseEulerFoam`, si ottiene un profilo più ampio ma si raggiungono valori di velocità al centro della colonna di 0.3 m/s, mentre per il `multiphaseEulerFoam` il profilo è delineato da un picco più stretto e alto, con valori tre volte più grandi al centro della colonna.

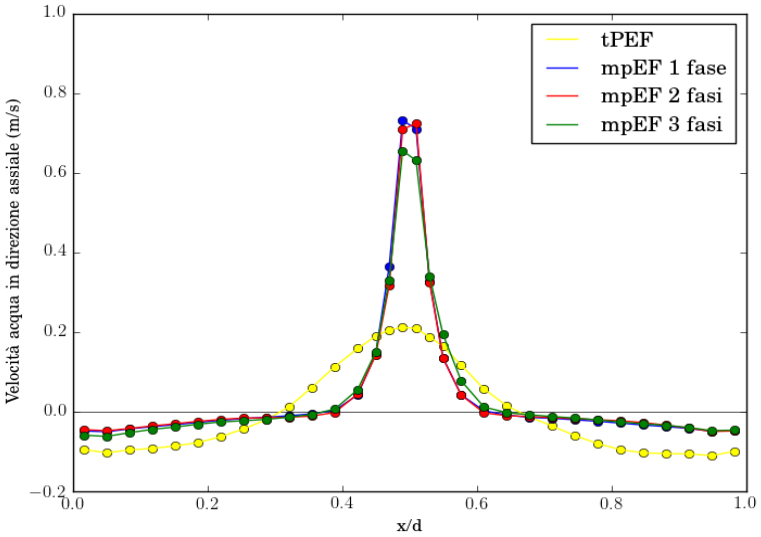

Figura 5.3: Profili di velocità assiale dell'acqua, mediati nel tempo, in funzione della distanza lungo l'asse x normalizzata rispetto alla larghezza della colonna, per le simulazioni con il `twoPhaseEulerFoam` e il `multiphaseEulerFoam`.

In figura 5.3 sono mostrati i profili di velocità dell'acqua in funzione della distanza lungo l'asse x normalizzata rispetto alla larghezza della colonna. Anche in questo caso, il profilo di velocità dell'acqua realizzato con il `twoPhaseEulerFoam` è sostanzialmente differente da quello prodotto con il multiphaseEulerFoam. Gli andamenti delle velocità dell'acqua sono simili a quelli dell'aria, ad eccezione delle zone vicine alle pareti in cui, con il `twoPhaseEulerFoam`, si raggiungono valori inferiori rispetto a quelli ottenuti con l'altro codice. Questo vuol dire che il `twoPhaseEulerFoam` è in grado di cogliere le zone di ricircolo di acqua ai lati della colonna, mentre lo stesso non si può dire del `multiphaseEulerFoam`.
Inoltre si è analizzato l'andamento dell'hold-up di aria in funzione del tempo di simulazione, pari a 100 s, delineato in figura 5.4. Come si può vedere la media volumica della frazione di fase dispersa, dopo un periodo transitorio che dura all'incirca 10 s, ha un andamento che oscilla attorno un valore medio: nella simulazione con `twoPhaseEulerFoam` l'ampiezza delle oscillazioni è minore rispetto a quelle con il `multiphaseEulerFoam`.
In tabella 5.1 sono riportati i valori medi di hold-up di aria all'interno della colonna, dopo il transitorio di 10 s. I valori ottenuti per le tre simulazioni con il `multiphaseEulerFoam` sono molto simili e si aggirano attorno al valore di 0.0158, per il `twoPhaseEulerFoam`, invece, si raggiunge un valore medio più piccolo e pari a 0.0114.

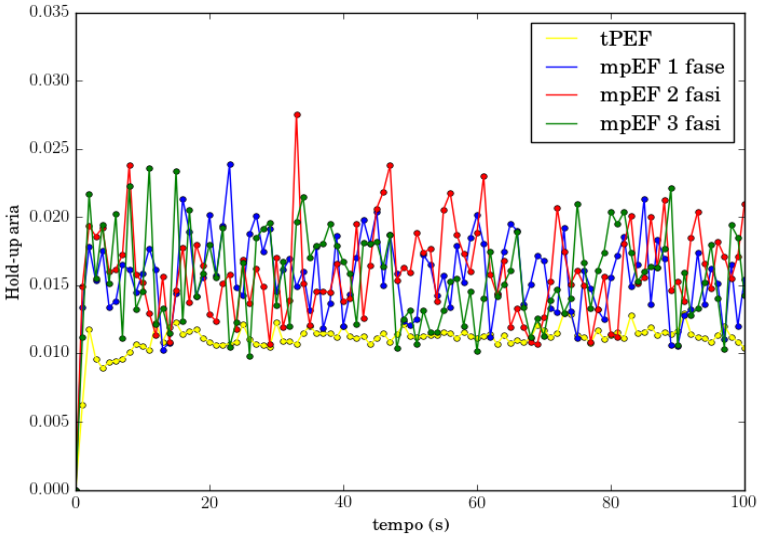

Figura 5.4: Andamenti dell'hold-up di aria in funzione del tempo, per le simulazioni con il `twoPhaseEulerFoam` e il `multiphaseEulerFoam`.

	twoPhaseEulerFoam	**multiphaseEulerFoam**		
	1 fase	1 fase	2 fasi	3 fasi
Hold-up aria	0.0114	0.0158	0.0159	0.0156

Tabella 5.1: Valori medi dell'hold-up di aria, dopo il transitorio, per le 4 simulazioni a seconda del numero di fasi disperse.

Dai confronti preliminari effettuati, è possibile dedurre che il `multiphaseEulerFoam` non permette di raggiungere risultati simili a quelli ottenuti con `twoPhaseEulerFoam`, pertanto le modifiche eseguite sui codici e le successive simulazioni effettuate sono volte alla verifica dell'incidenza delle differenze presenti tra i due codici.

5.2 Risultati delle modifiche sui codici

Di seguito verranno discussi i risultati derivati dalle modifiche effettuate sul codice `twoPhaseEulerFoam` e sul `multiphaseEulerFoam`, realizzando dei confronti tra i codici stessi, e con i risultati ottenuti con i codici iniziali discussi nel paragrafo precedente.

5.2.1 `TwoPhaseEulerFoam`

Operando le modifiche riassunte nel paragrafo 3.4.1, è possibile analizzare le differenze tra il codice `twoPhaseEulerFoam` e quello modificato.
In figura 5.5 vengono rappresentati i profili delle frazioni volumiche dell'aria, mediati nel tempo escludendo il transitorio iniziale di 10 s, in funzione della distanza lungo l'asse x normalizzata rispetto alla larghezza della colonna, ad un'altezza pari alla metà del livello di liquido: in giallo viene indicato l'andamento per il codice `twoPhaseEulerFoam` (tPEF), mentre in viola quello per il `twoPhaseEulerFoam` modificato (tPEF modificato). Le modifiche effettuate sul codice sono relative al termine di drag, poiché, come si è visto nel paragrafo 3.4.1, non è direttamente confrontabile con quello del `multiphaseEulerFoam`. Si ricorda che nella forza di trascinamento si è introdotta la frazione volumica dell'acqua.

Le due curve risultano simili ad eccezione del picco raggiunto al centro della colonna, che nel caso del codice modificato raggiunge valori leggermente inferiori rispetto al codice `twoPhaseEulerFoam` iniziale, perché, come illustrato in figura 3.3, introducendo la frazione volumica dell'acqua nel termine di drag si raggiungono valori inferiori della frazione volumica di aria.

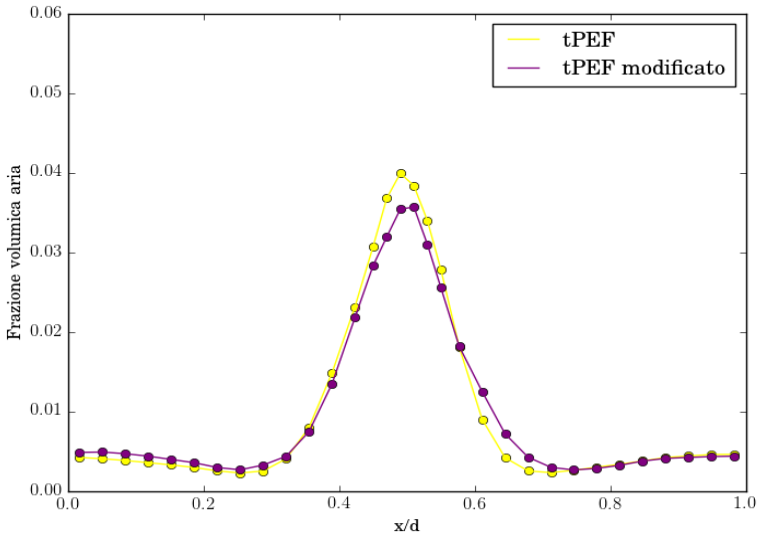

Figura 5.5: Profili della frazione volumica di aria, mediati nel tempo, in funzione della distanza lungo l'asse x normalizzata rispetto alla larghezza della colonna, per le simulazioni con il `twoPhaseEulerFoam` e il `twoPhaseEulerFoam` modificato.

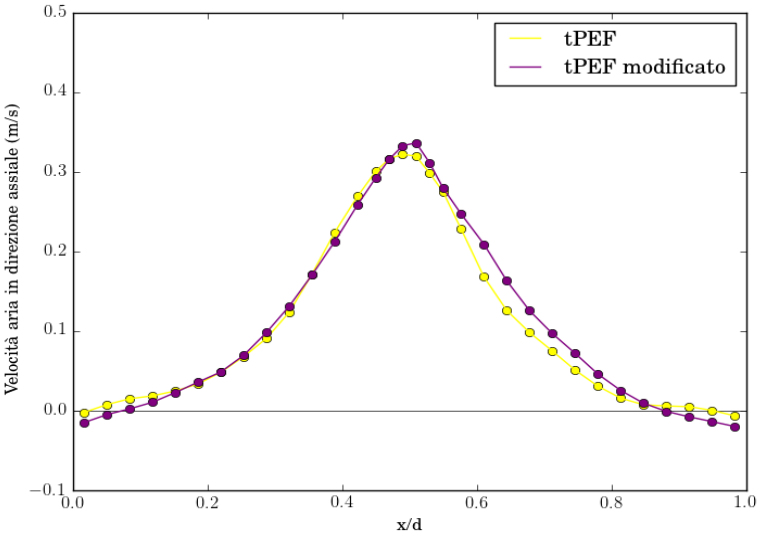

Figura 5.6: Profili di velocità assiale dell'aria, mediati nel tempo, in funzione della distanza lungo l'asse x normalizzata rispetto alla larghezza della colonna, per le simulazioni con il `twoPhaseEulerFoam` e il `twoPhaseEulerFoam` modificato.

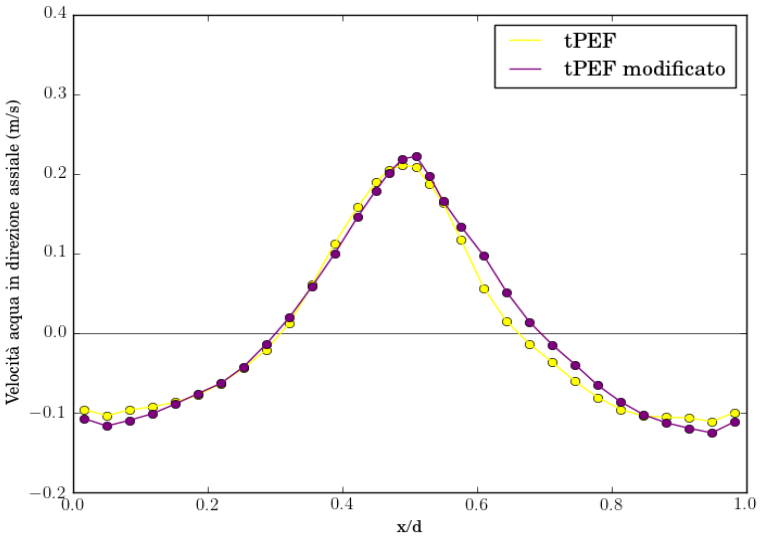

Figura 5.7: Profili di velocità assiale dell'acqua, mediati nel tempo, in funzione della distanza lungo l'asse x normalizzata rispetto alla larghezza della colonna, per le simulazioni con il `twoPhaseEulerFoam` e il `twoPhaseEulerFoam` modificato.

I profili di velocità dell'aria, mediati nel tempo escludendo il transitorio iniziale di 10 s, in funzione della distanza lungo l'asse x normalizzata rispetto alla larghezza della colonna, sono illustrati in figura 5.6, mentre i profili delle velocità dell'acqua sono mostrati in figura 5.7.
I profili di velocità appaiono quasi del tutto coincidenti ad eccezione della zona di x/d tra 0.6 e 0.8, dove i valori di velocità per il `twoPhaseEulerFoam` modificato raggiungono valori leggermente più alti, questo può essere dovuto al fatto che i profili del pennacchio di bolle potrebbero non essere stati mediati per tempi sufficientemente lunghi.

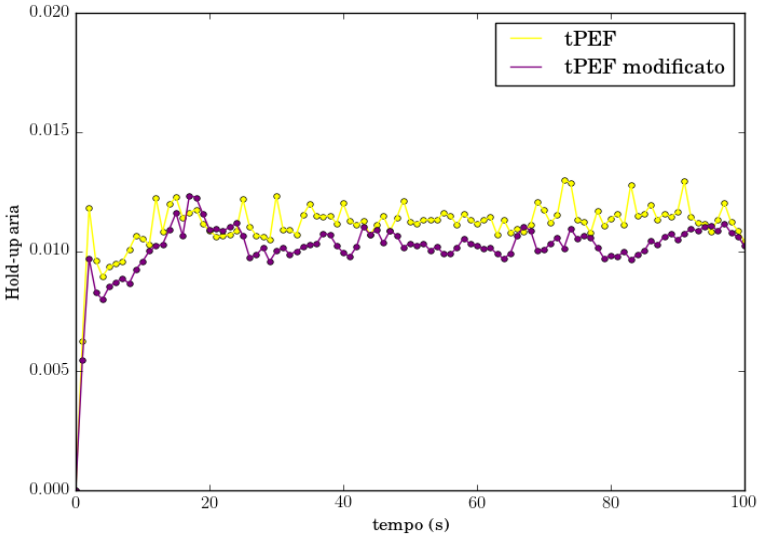

Figura 5.8: Andamenti dell'hold-up di aria in funzione del tempo, per le simulazioni con il `twoPhaseEulerFoam` e il `twoPhaseEulerFoam` modificato.

Infine si sono studiati gli andamenti dell'hold-up di aria in funzione del tempo di simulazione, delineato in figura 5.8. Il valore dell'hold-up medio di aria, dopo il transitorio di 10 s, per il `twoPhaseEulerFoam`, come già detto nel paragrafo precedente, è pari a 0.0114, mentre quello per il codice modificato è pari a 0.0104, perché si raggiungono valori di frazione volumica di aria inferiori rispetto ai risultati ottenuti con il codice non modificato.
Dai risultati ottenuti, è possibile notare che le modifiche effettuate sul `twoPhaseEulerFoam` sono state marginali sia per quanto riguarda i profili di velocità e di frazione volumica, sia per gli andamenti di hold-up di aria.

5.2.2 `MultiphaseEulerFoam`

Di seguito vengono mostrati i risultati relativi alle modifiche effettuate nel codice `multiphaseEulerFoam`, riassunte nel paragrafo 3.4.2.
In figura 5.9 vengono rappresentati i profili di frazione volumica dell'aria, mediati nel tempo escludendo il transitorio iniziale di 10 s, in funzione della distanza lungo l'asse x normalizzata rispetto alla larghezza della colonna (d), ad un'altezza pari alla metà del livello di liquido.

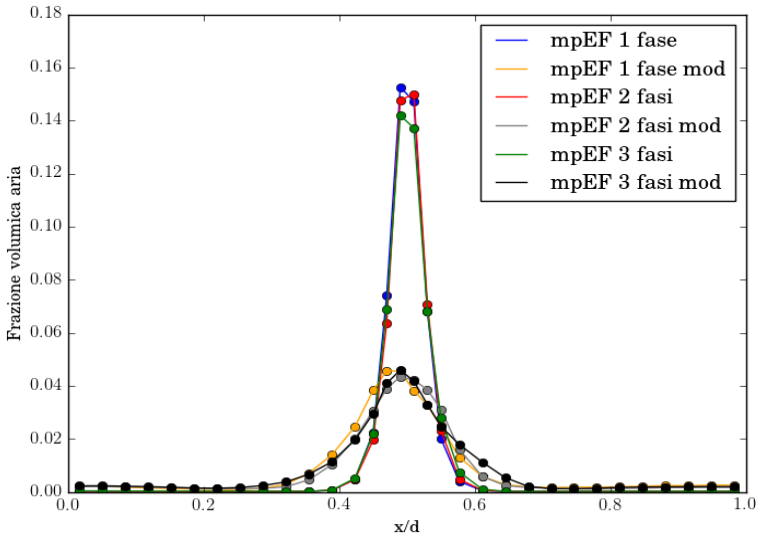

Figura 5.9: Profili di frazione volumica dell'aria, mediati nel tempo, in funzione della distanza lungo l'asse x normalizzata rispetto alla larghezza della colonna, per le simulazioni con il `multiphaseEulerFoam` e il `multiphaseEulerFoam` modificato.

In blu, rosso e verde vengono indicati rispettivamente gli andamenti per 1 fase dispersa, per 2 fasi disperse e per 3 fasi disperse con il codice `multiphaseEulerFoam`, già mostrati nel paragrafo 5.1, mentre in arancione, grigio e nero vengono rappresentati rispettivamente gli andamenti per 1 fase dispersa (mpEF 1 fase mod), per 2 fasi disperse (mpEF 2 fasi mod) e per 3 fasi disperse (mpEF 3 fasi mod) con il codice modificato.
Come visto nel paragrafo 3.4.2, la prima modifica effettuata nel `multiphaseEulerFoam` è stata l'introduzione del termine `residualAlpha` nelle matrici dei coefficienti delle velocità `A` e alle matrici dei termini noti `H`, mentre la seconda modifica è relativa al set di equazioni per la frazione volumica delle fasi disperse, che è stato completamente riscritto, poiché l'implementazione del modello multifluido in `multiphaseEulerFoam` include

termini aggiuntivi che fanno riferimento alla dinamica dell'interfaccia fra le fasi che non sono presenti in `twoPhaseEulerFoam`.
Per quanto riguarda la frazione volumica, è possibile notare che nel caso del codice modificato si raggiungono valori decisamente inferiori rispetto a quelli con il codice `multiphaseEulerFoam` iniziale. In particolare si ha un picco che arriva a valori di circa 0.15 per il codice iniziale, mentre si riscontrano valori della frazione volumica 3 volte inferiori per il codice modificato. Questa differenza è dovuta sia all'introduzione del termine `residualAlpha` che viene utilizzato quando la frazione volumica tende a zero, sia all'eliminazione dei termini di curvatura interfacciale.
Inoltre è importante notare come, anche con il codice modificato, si ottengono profili coincidenti per tutte le fasi disperse presenti, come ci si aspetta.
I profili di velocità dell'aria, dopo il transitorio di 10 s, in funzione della distanza lungo l'asse x normalizzata rispetto alla larghezza della colonna, sono illustrati in figura 5.10.

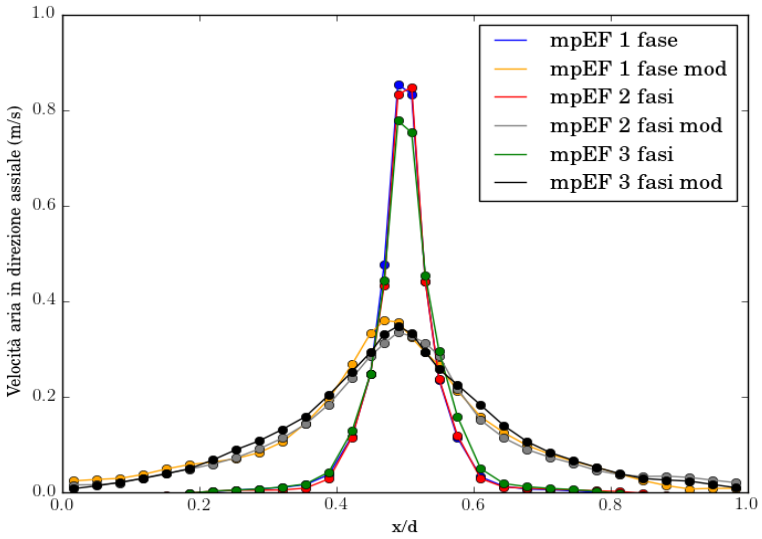

Figura 5.10: Profili di velocità assiale dell'aria, mediati nel tempo, in funzione della distanza lungo l'asse x normalizzata rispetto alla larghezza della colonna, per le simulazioni con il `multiphaseEulerFoam` e il `multiphaseEulerFoam` modificato.

Anche per quanto riguarda i profili di velocità dell'aria con il codice modificato, si riscontra una diminuzione dei valori di velocità al centro della colonna rispetto a quelli ottenuti con il codice iniziale, in modo coerente con quanto avviene per i profili di frazione volumica, a causa delle modifiche effettuate nel codice.
Inoltre si può notare come il profilo per il codice modificato sia molto più ampio rispetto al codice iniziale.
In figura 5.11 sono mostrati i profili di velocità dell'acqua, dopo il transitorio di 10 s, in funzione della distanza lungo l'asse x normalizzata rispetto alla larghezza della colonna. Per quanto riguarda la velocità dell'acqua, si riscontrano gli stessi cambiamenti del profilo di velocità dell'aria, con la differenza che, vicino alle pareti della colonna, le velocità dell'acqua per il codice modificato assumono valori inferiori rispetto al codice iniziale. Questo vuol dire che le modifiche effettuate hanno permesso di cogliere le zone di ricircolo di acqua ai lati della colonna.

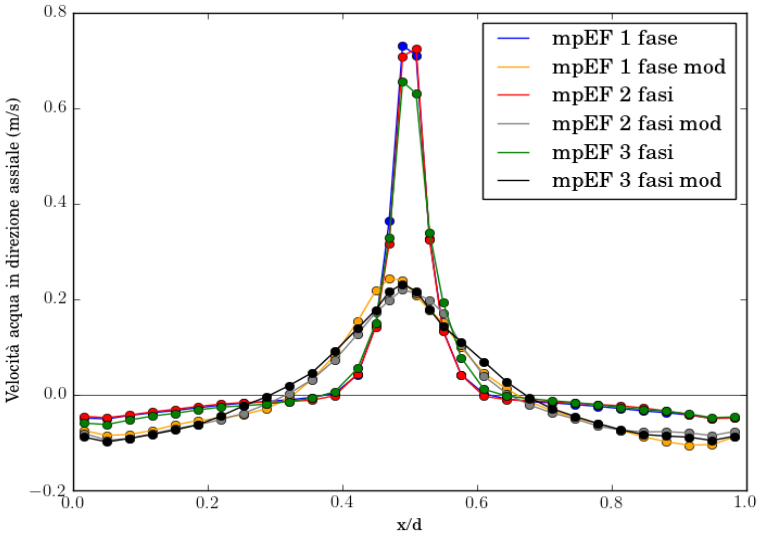

Figura 5.11: Profili di velocità assiale dell'acqua, mediati nel tempo, in funzione della distanza lungo l'asse x normalizzata rispetto alla larghezza della colonna, per le simulazioni con il `multiphaseEulerFoam` e il `multiphaseEulerFoam` modificato.

Gli andamenti dell'hold-up di aria in funzione del tempo di simulazione sono delineati in figura 5.12.

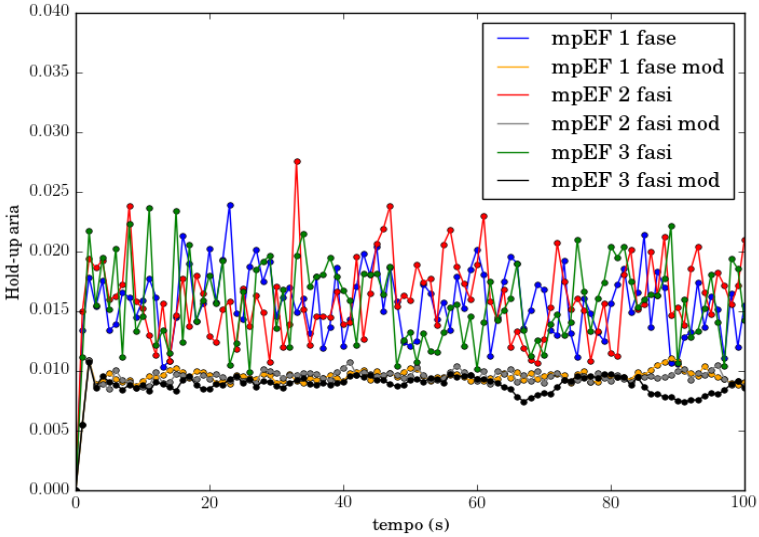

Figura 5.12: Andamenti dell'hold-up di aria in funzione del tempo, per le simulazioni con il `multiphaseEulerFoam` e il `multiphaseEulerFoam` modificato.

In tabella 5.2 vengono indicati i valori medi di hold-up di aria, dopo il transitorio di 10 s, in funzione del tempo di simulazione, per tutte le simulazioni effettuate con il `multiphaseEulerFoam`.
L'hold-up di aria raggiunge, quindi, valori più piccoli con le simulazioni ottenute con il codice modificato, perché si raggiungono valori di frazione volumica di aria inferiori rispetto ai risultati ottenuti con il codice non modificato.

Dai risultati ottenuti, è emerso che le modifiche effettuate sul `multiphaseEulerFoam` hanno avuto una notevole influenza sia sui profili di velocità e di frazione volumica, sia sugli andamenti di hold up di aria.

Hold-up aria	**multiphaseEulerFoam**		
	1 fase	2 fasi	3 fasi
mpEF	0.0160	0.0161	0.0158
mpEF mod	0.0096	0.0095	0.0088

Tabella 5.2: Valori medi dell'hold-up di aria, dopo il transitorio, per le simulazioni con il `multiphaseEulerFoam` e il `multiphaseEulerFoam` modificato.

5.2.3 Confronto tra i codici modificati

Di seguito vengono mostrati i risultati ottenuti confrontando le modifiche del `twoPhaseEulerFoam` con quelle del `multiphaseEulerFoam`. In questo modo, come visto nel paragrafo 3.4.2, il confronto tra i codici è coerente.
In figura 5.13 sono mostrati i profili di frazione volumica di aria, mediati nel tempo escludendo il transitorio iniziale di 10 s, in funzione della distanza lungo l'asse x normalizzata rispetto alla larghezza della colonna, per le simulazioni con il `twoPhaseEulerFoam` e `multiphaseEulerFoam` modificati.

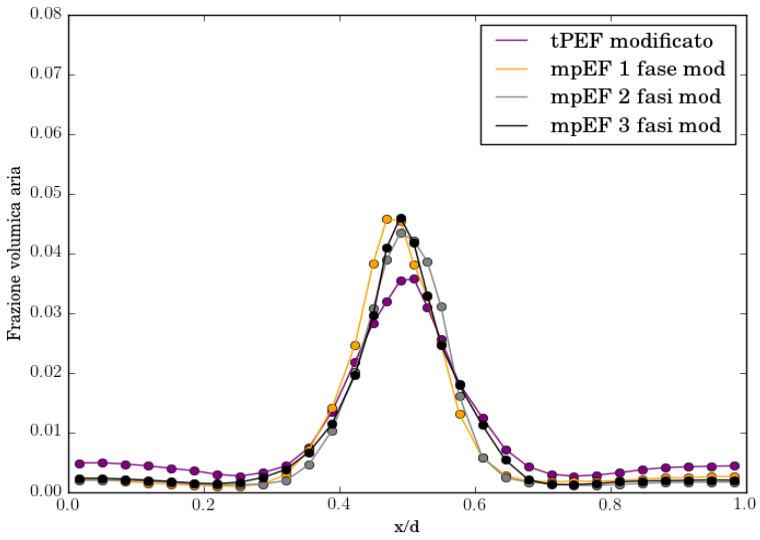

Figura 5.13: Profili di frazione volumica di aria, mediati nel tempo, in funzione della distanza lungo l'asse x normalizzata rispetto alla larghezza della colonna, per le simulazioni con il `twoPhaseEulerFoam` e `multiphaseEulerFoam` modificati.

Si può facilmente notare come, mediante le modifiche effettuate, gli andamenti delle frazioni volumiche sono molto più simili rispetto a quelli ottenuti con i codici senza modifiche. Il valore del picco della frazione volumica al centro della colonna è pari a 0.035 per il `twoPhaseEulerFoam`, mentre per il `multiphaseEulerFoam` si aggira attorno a 0.05. Ai lati con il `twoPhaseEulerFoam` si raggiungono valori più alti rispetto al `multiphaseEulerFoam`, questo significa che il valore integrale è simile: per il `twoPhaseEulerFoam` è di circa 0.0110, mentre per il `multiphaseEulerFoam` è di 0.0108.

I profili di velocità dell'aria, mediati nel tempo escludendo il transitorio iniziale di 10 s, in funzione della lunghezza normalizzata sono illustrati in figura 5.14.

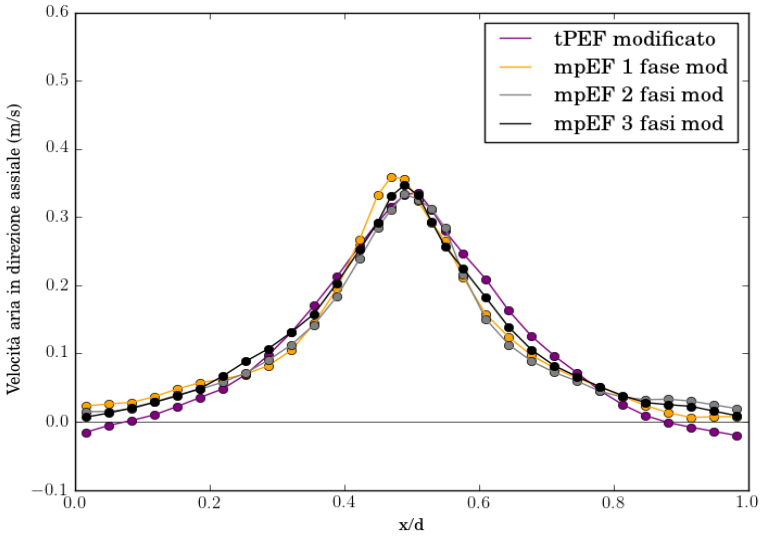

Figura 5.14: Profili di velocità assiale dell'aria, mediati nel tempo, in funzione della distanza lungo l'asse x normalizzata rispetto alla larghezza della colonna, per le simulazioni con il `twoPhaseEulerFoam` e `multiphaseEulerFoam` modificati.

I profili di velocità dell'aria sono quasi del tutto coincidenti e non si riscontrano differenze sostanziali nell'utilizzo dei due codici modificati.
In figura 5.15 sono mostrati i profili di velocità dell'acqua, mediati nel tempo escludendo il transitorio iniziale di 10 s, in funzione di x/d.

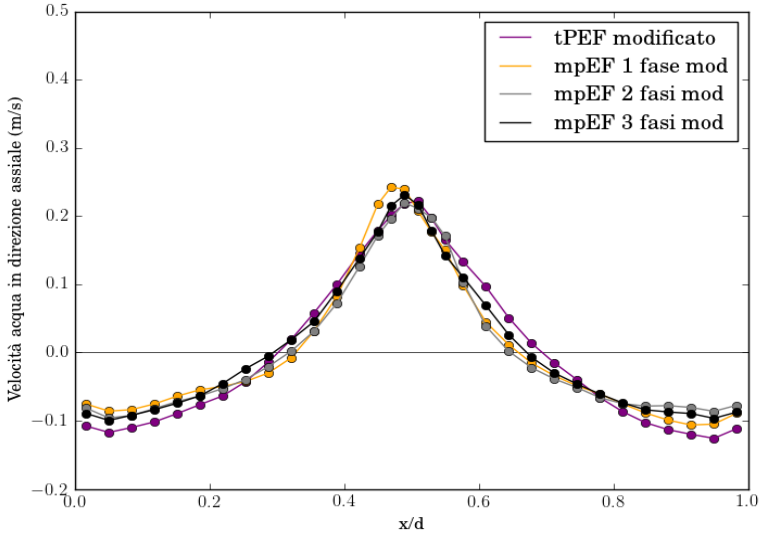

Figura 5.15: Profili di velocità assiale dell'acqua, mediati nel tempo, in funzione della distanza lungo l'asse x normalizzata rispetto alla larghezza della colonna, per le simulazioni con il `twoPhaseEulerFoam` e `multiphaseEulerFoam` modificati.

Anche in questo caso i profili di velocità dell'acqua risultano quasi del tutto identici

fra loro, perché le modifiche effettuate hanno permesso di cogliere le zone di ricircolo di acqua ai lati della colonna, che prima non venivano rilevate.

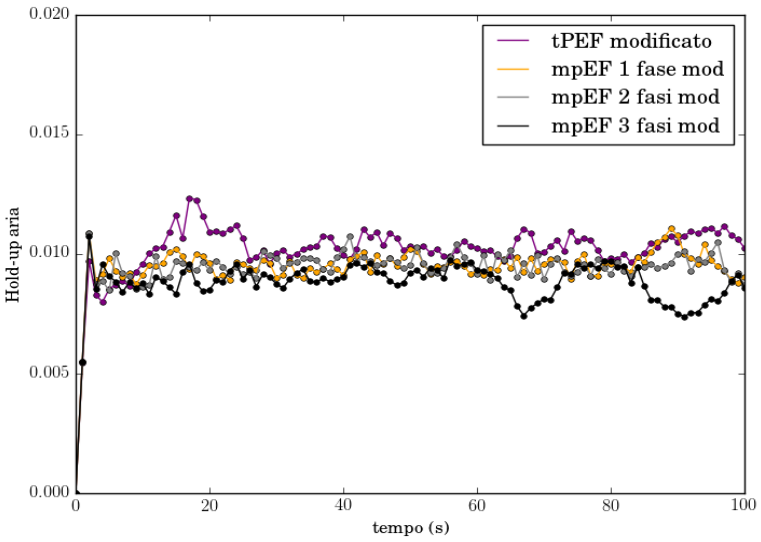

Figura 5.16: Andamenti dell'hold-up di aria in funzione del tempo, per le simulazioni con il `twoPhaseEulerFoam` e `multiphaseEulerFoam` modificati.

Gli andamenti dell'hold-up di aria in funzione del tempo di simulazione sono delineati in figura 5.16. In tabella 5.3 vengono mostrati i valori medi di hold-up di aria, dopo il transitorio di 10 s.
Si può facilmente notare come l'hold-up medio di aria derivato dal `twoPhaseEulerFoam` sia molto più simile ai valori ottenuti con il `multiphaseEulerFoam` modificato rispetto a quelli con il `multiphaseEulerFoam` iniziale, perché si raggiungono valori di frazione volumica di aria simili fra i due codici.

	twoPhaseEulerFoam	**multiphaseEulerFoam**		
	1 fase	1 fase	2 fasi	3 fasi
Hold-up aria	0.0104	0.0096	0.0095	0.0088

Tabella 5.3: Valori medi dell'hold-up di aria, dopo il transitorio, per le simulazioni con il `twoPhaseEulerFoam` e `multiphaseEulerFoam` modificati, a seconda del numero di fasi disperse.

Infine, si sono effettuati degli ulteriori confronti tra i due codici modificati, utilizzando un diametro per le bolle d'aria di 10 mm .
In figura 5.17 sono mostrati i profili di frazione volumica dell'aria, mediati nel tempo escludendo il transitorio iniziale di 10 s, in funzione della distanza lungo l'asse x normalizzata rispetto alla larghezza della colonna, ottenuti utilizzando un diametro di 10 mm per le bolle d'aria.
Nei casi in cui sono presenti più fasi secondarie, si è utilizzato lo stesso diametro per ogni singola fase dispersa.

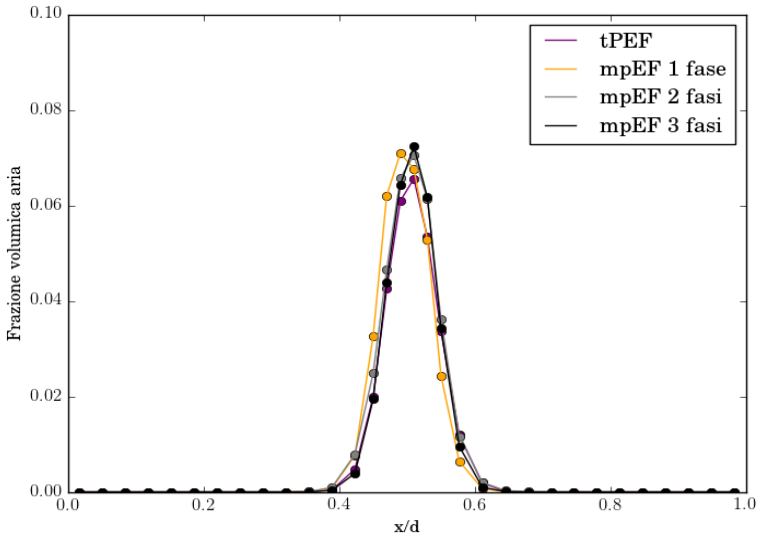

Figura 5.17: Profili di frazione volumica dell'aria, mediati nel tempo, in funzione di x/d, per le simulazioni con il `twoPhaseEulerFoam` e `multiphaseEulerFoam` modificati, utilizzando un diametro delle bolle d'aria di 10 mm.

I profili di frazione volumica per tutte le simulazioni risultano quasi coincidenti, e raggiungono valori pari a 0.07, al centro della colonna, leggermente più alti rispetto ai casi simulati con un diametro delle bolle di 1 mm, in cui si arrivava ad un massimo di 0.05.
I profili di velocità dell'aria, mediati nel tempo escludendo il transitorio iniziale di 10 s, in funzione della lunghezza normalizzata sono illustrati in figura 5.18.

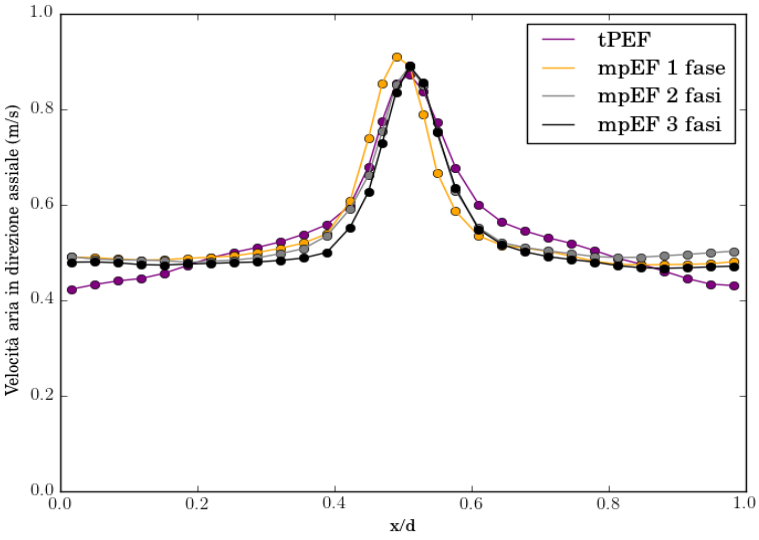

Figura 5.18: Profili di velocità assiale dell'aria, mediati nel tempo, in funzione di x/d, per le simulazioni con il `twoPhaseEulerFoam` e `multiphaseEulerFoam` modificati, utilizzando un diametro delle bolle d'aria di 10 mm.

Si può dedurre che utilizzando un diametro delle bolle maggiore, si ottengono dei

profili di velocità traslati a valori maggiori rispetto ai casi con diametri delle bolle più piccoli, perché bolle di dimensioni diverse si muovono con velocità differenti.
In figura 5.19 sono mostrati i profili di velocità dell'acqua, mediati nel tempo escludendo il transitorio iniziale di 10 s, in funzione della lunghezza normalizzata.

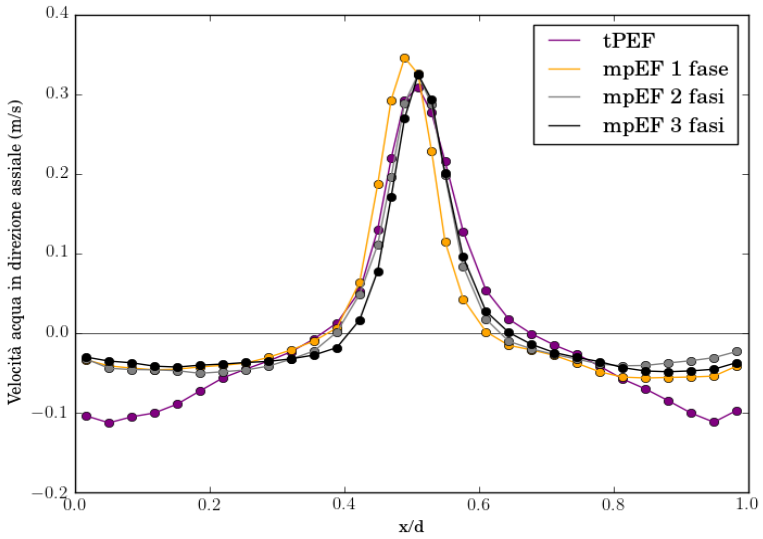

Figura 5.19: Profili di velocità assiale dell'acqua, mediati nel tempo, in funzione di x/d, per le simulazioni con il `twoPhaseEulerFoam` e `multiphaseEulerFoam` modificati, utilizzando un diametro delle bolle d'aria di 10 mm.

I profili di velocità dell'acqua risultano quasi del tutto coincidenti per i casi simulati con il `multiphaseEulerFoam`, mentre quello ottenuto con il `twoPhaseEulerFoam` si differenzia nelle zone adiacenti alle pareti, in cui assume valori di velocità inferiori. In questo caso il `multiphaseEulerFoam` non ha permesso di cogliere le zone di ricircolo di acqua ai lati della colonna con lo stesso grado di accuratezza del `twoPhaseEulerFoam`.
Gli andamenti dell'hold-up di aria in funzione del tempo di simulazione sono delineati in figura 5.20. In tabella 5.4 vengono mostrati i valori medi di hold-up di aria, dopo il transitorio di 10 s.
Utilizzando diametri delle bolle pari a 10 mm, i valori medi di hold-up di aria risultano più simili fra loro rispetto ai casi con diametri delle bolle pari a 1 mm, perché i valori di frazione volumica raggiunti dalle simulazioni con un diametro delle bolle maggiore, sono molto più simili fra loro rispetto a quelle ottenute con diametri minori.

	twoPhaseEulerFoam	**multiphaseEulerFoam**		
	1 fase	1 fase	2 fasi	3 fasi
Hold-up aria	0.0044	0.0047	0.0047	0.0047

Tabella 5.4: Valori medi dell'hold-up di aria, dopo il transitorio, per le 4 simulazioni a seconda del numero di fasi disperse, utilizzando un diametro delle bolle d'aria di 10 mm.

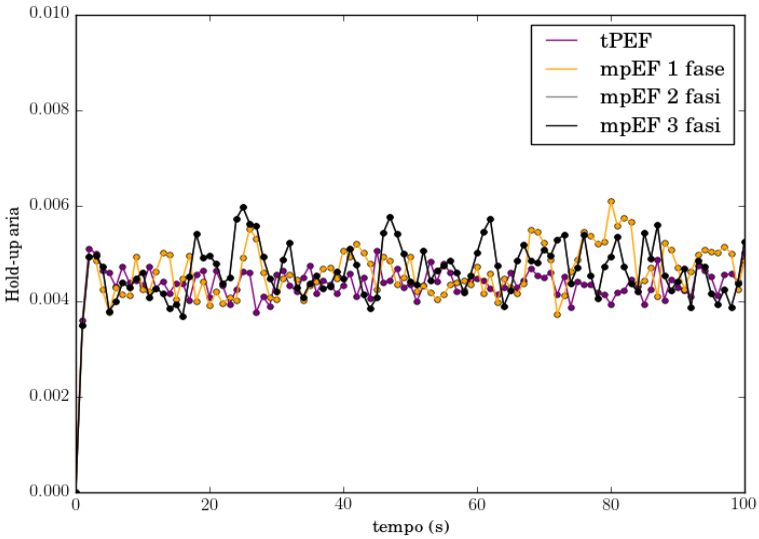

Figura 5.20: Andamenti dell'hold-up di aria in funzione del tempo, per le simulazioni con il `twoPhaseEulerFoam` e `multiphaseEulerFoam` modificati, utilizzando un diametro delle bolle d'aria di 10 *mm*.

In conclusione è possibile affermare che il modello multifluido implementato nel codice `multiphaseEulerFoam` non è equivalente al modello Two-Fluid implementato nel `twoPhaseEulerFoam`.
In particolare le differenze più significative fra i due codici sono dovute alle diverse modalità di implementazione delle equazioni di trasporto della frazione volumica, infatti, come già spiegato nel paragrafo 3.3, nel caso del `twoPhaseEulerFoam` il termine convettivo viene opportunamente scomposto per evitare di incorrere in problemi legati alla limitatezza della frazione volumica, mentre nel `multiphaseEulerFoam` i termini convettivi vengono risolti così come presentati nell'equazione di conservazione della frazione volumica e sono presenti dei termini di curvatura interfacciale che invece non sono presenti nell'altro codice.
Pertanto si è dimostrato che, modificando il modello multifluido implementato nel `multiphaseEulerFoam`, come descritto nel paragrafo 3.4.2, è possibile ottenere dei risultati coerenti con quelli conseguiti con il modello Two-Fluid del codice `twoPhaseEulerFoam`.

Capitolo 6

Conclusioni

Nel corso del presente lavoro è stato analizzato il solutore `multiphaseEulerFoam`, appartenente al codice libero di fluidodinamica computazionale OpenFOAM, applicato ad un sistema gas-liquido in una colonna a bolle, caratterizzato dalla presenza di bolle d'aria disperse in una fase continua.
Per verificare l'accuratezza del codice `multiphaseEulerFoam`, sono stati investigati tre casi equivalenti rispettivamente con 1, 2 e 3 fasi disperse, e si sono poi confrontati i risultati con un altro solutore, il `twoPhaseEulerFoam`, già utilizzato in precedenza per questo tipo di sistemi [17].
Si è, quindi, esaminata la capacità del solutore `multiphaseEulerFoam` di risolvere sistemi multifase con il modello multifluido implementato.
L'analisi è stata condotta su una colonna a bolle di sezione rettangolare, adoperando caratteristiche e condizioni operative già proposte negli articoli di Diaz et al. [9] e di Buffo et al. [5].
Inoltre, sono state condotte simulazioni con un diametro delle bolle d'aria maggiore rispetto ai casi inizialmente esaminati, in modo da valutare l'influenza di questo parametro.
Confronti preliminari tra i due solutori hanno dimostrato che il `multiphaseEulerFoam` non consente di raggiungere il grado di accuratezza ottenuto con il `twoPhaseEulerFoam`, a causa delle notevoli differenze tra i due codici.
La prima differenza è stata riscontrata nell'implementazione delle equazioni che governano il modello multifluido, e la seconda nell'equazione della forza di drag per unità di volume.
Pertanto, è stato necessario effettuare modifiche su entrambi i codici, volte alla verifica dell'incidenza delle differenze presenti tra i due solutori sui risultati.
Nel `twoPhaseEulerFoam` si è adoperata una correzione della forza di drag che tenesse conto della frazione volumica dell'acqua, mentre nel `multiphaseEulerFoam` si sono modificate le equazioni relative al trasporto della frazione volumica delle fasi disperse e quelle utilizzate per la costruzione dell'equazione della pressione.
Dalle simulazioni svolte con le modifiche sul `twoPhaseEulerFoam` non sono emerse variazioni significative nei profili di velocità, della frazione volumica e di hold up di aria. Per quanto riguarda le modifiche sul `multiphaseEulerFoam`, si sono riscontrati cambiamenti sostanziali nei profili, che hanno fornito risultati migliori rispetto a quelli ottenuti con il codice non modificato.
I risultati di tutte le simulazioni svolte hanno quindi confermato come l'implementazione del modello multifluido nel codice `multiphaseEulerFoam` dia risultati sostanzialmente diversi rispetto al `twoPhaseEulerFoam`. Ciò è causato dalle differenze fra i due codici, in particolar modo alle diverse modalità di implementazione delle equazioni di trasporto

della frazione volumica delle fasi disperse, come visto nel paragrafo 3.3.
In conclusione è possibile affermare che il modello multifluido utilizzato nel codice `multiphaseEulerFoam` presenta ancora dei limiti nella simulazione di sistemi dispersi gas-liquido.
Tuttavia si è dimostrato che, attraverso opportune modifiche al modello multifluido implementato nel `multiphaseEulerFoam`, è possibile ottenere dei risultati accurati e congruenti con il modello Two-Fluid del codice `twoPhaseEulerFoam`.
Gli sviluppi futuri di questo studio potrebbero riguardare la simulazione di sistemi polidispersi, in cui ciascuna fase dispersa sia composta da elementi con dimensione caratteristica diversa.

Appendice A

TwoPhaseEulerFoam

Tutte le appendici che seguiranno, sono degli estratti del codice `twoPhaseEulerFoam` della versione 5 di OpenFOAM. Le linee di codice complete sono reperibili sul sito https://openfoam.org. I titoli delle appendici sono i nomi dei file principali che compongono il solver `twoPhaseEulerFoam`.

A.1 TwoPhaseSystem.C

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

void Foam::twoPhaseSystem::solve()
{
 volScalarField& alpha1 = phase1_;
 volScalarField& alpha2 = phase2_;

 const surfaceScalarField& phi1 = phase1_.phi();
 const surfaceScalarField& phi2 = phase2_.phi();

 label nAlphaSubCycles(readLabel(alphaControls.lookup("nAlphaSubCycles")));
 label nAlphaCorr(readLabel(alphaControls.lookup("nAlphaCorr")));

 word alphaScheme("div(phi," + alpha1.name() + ')');
 word alpharScheme("div(phir," + alpha1.name() + ')');

 alpha1.correctBoundaryConditions();

 surfaceScalarField phic("phic", phi_);
 surfaceScalarField phir("phir", phi1 - phi2);

 for (int acorr=0; acorr<nAlphaCorr; acorr++)
 {
   volScalarField::Internal Sp
   (IOobject
   ("Sp",
   runTime.timeName(),
   mesh_),
   mesh_,
```

```
 dimensionedScalar("Sp", dgdt_.dimensions(), 0.0));

 volScalarField::Internal Su
 (IOobject
 ("Su",
 runTime.timeName(),
 mesh_),
 fvc::div(phi_)*min(alpha1, scalar(1)));

 forAll(dgdt_, celli)
 { if (dgdt_[celli] > 0.0)
   {
    Sp[celli] -= dgdt_[celli]/max(1.0 - alpha1[celli], 1e-4);
    Su[celli] += dgdt_[celli]/max(1.0 - alpha1[celli], 1e-4);
   }
    else if (dgdt_[celli] < 0.0)
   {
    Sp[celli] += dgdt_[celli]/max(alpha1[celli], 1e-4);
    }
 }

 surfaceScalarField alphaPhic1
    (
      fvc::flux
      (phic,
       alpha1,
       alphaScheme)
      + fvc::flux
       (-fvc::flux(-phir, scalar(1) - alpha1, alpharScheme),
        alpha1,
        alpharScheme)
     );
phase1_.correctInflowOutflow(alphaPhic1);

 if (nAlphaSubCycles > 1)
 {
   for(subCycle<volScalarField> alphaSubCycle(alpha1, nAlphaSubCycles);
    !(++alphaSubCycle).end();
    {
    surfaceScalarField alphaPhic10(alphaPhic1);

     MULES::explicitSolve
     (
      geometricOneField(),
      alpha1,
      phi_,
      alphaPhic10,
     (alphaSubCycle.index()*Sp)(),
     (Su - (alphaSubCycle.index() - 1)*Sp*alpha1)(),
```

```
        phase1_.alphaMax(),
        0
        );

       if (alphaSubCycle.index() == 1)
       {phase1_.alphaPhi() = alphaPhic10;}
       else
       {phase1_.alphaPhi() += alphaPhic10;}
      }

    phase1_.alphaPhi() /= nAlphaSubCycles;
    }
  else
  {MULES::explicitSolve
    (
     geometricOneField(),
     alpha1,
     phi_,
     alphaPhic1,
     Sp,
     Su,
     phase1_.alphaMax(),
     0
    );
  phase1_.alphaPhi() = alphaPhic1;
  }

  phase1_.alphaRhoPhi() =
  fvc::interpolate(phase1_.rho())*phase1_.alphaPhi();

  phase2_.alphaPhi() = phi_ - phase1_.alphaPhi();
  phase2_.correctInflowOutflow(phase2_.alphaPhi());
  phase2_.alphaRhoPhi() =
  fvc::interpolate(phase2_.rho())*phase2_.alphaPhi();

  Info<< alpha1.name() << " volume fraction = "
  << alpha1.weightedAverage(mesh_.V()).value()
  << "  Min(" << alpha1.name() << ") = " << min(alpha1).value()
  << "  Max(" << alpha1.name() << ") = " << max(alpha1).value()
  << endl;

  alpha1.max(0);
  alpha1.min(1);
  alpha2 = scalar(1) - alpha1;
}
// ************************************************************************* //
```

A.2 UEqns.H

```
// ************************************************************************* //
U1Eqn =
(
  fvm::ddt(alpha1, rho1, U1) + fvm::div(alphaRhoPhi1, U1)
    - fvm::Sp(contErr1, U1) + MRF.DDt(alpha1*rho1 + Vm, U1)
  + phase1.turbulence().divDevRhoReff(U1)
==
- Vm*(fvm::ddt(U1)
    + fvm::div(phi1, U1)
    - fvm::Sp(fvc::div(phi1), U1)
     - DDtU2
     )
+ fvOptions(alpha1, rho1, U1)
);
U1Eqn.relax();
U1Eqn += fvm::Sp(Kd, U1);
fvOptions.constrain(U1Eqn);
U1.correctBoundaryConditions();
fvOptions.correct(U1);

U2Eqn =
(
     fvm::ddt(alpha2, rho2, U2) + fvm::div(alphaRhoPhi2, U2)
     - fvm::Sp(contErr2, U2) + MRF.DDt(alpha2*rho2 + Vm, U2)
    + phase2.turbulence().divDevRhoReff(U2)
==
- Vm*(fvm::ddt(U2)
     + fvm::div(phi2, U2)
    - fvm::Sp(fvc::div(phi2), U2)
     - DDtU1
        )
+ fvOptions(alpha2, rho2, U2)
);
U2Eqn.relax();
U2Eqn += fvm::Sp(Kd, U2);
fvOptions.constrain(U2Eqn);
U2.correctBoundaryConditions();
fvOptions.correct(U2);

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

A.3 DragModel.C

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

Foam::tmp<Foam::volScalarField> Foam::dragModel::Ki() const
{
     return
     0.75
     *CdRe()
     *swarmCorrection_->Cs()
     *pair_.continuous().rho()
     *pair_.continuous().nu()
     /sqr(pair_.dispersed().d());
}

Foam::tmp<Foam::volScalarField> Foam::dragModel::K() const
{
  return max(pair_.dispersed(), pair_.dispersed().residualAlpha())*Ki();
}

Foam::tmp<Foam::surfaceScalarField> Foam::dragModel::Kf() const
{
  return
     max( fvc::interpolate(pair_.dispersed()),
     pair_.dispersed().residualAlpha())*fvc::interpolate(Ki());
}
// ************************************************************************* //
```

A.4 SchillerNaumann.C

```
// * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * //
 Foam::tmp<Foam::volScalarField>
 Foam::dragModels::SchillerNaumann::CdRe() const
{
  volScalarField Re(pair_.Re());
  return
  neg(Re - 1000)*24.0*(1.0 + 0.15*pow(Re, 0.687))
```

```
  + pos0(Re - 1000)*0.44* max(Re, residualRe_);
}
// ************************************************************************* //
```

A.5 PEqn.H

```
// ************************************************************************* //
surfaceScalarField alphaf1("alphaf1", fvc::interpolate(alpha1));
surfaceScalarField alphaf2("alphaf2", scalar(1) - alphaf1);

volScalarField rAU1
(
  IOobject::groupName("rAU", phase1.name()),
  1.0/(U1Eqn.A()+ max(phase1.residualAlpha() - alpha1, scalar(0))
     *rho1/runTime.deltaT())
);
volScalarField rAU2
(
  IOobject::groupName("rAU", phase2.name()),
  1.0/(U2Eqn.A()+ max(phase2.residualAlpha() - alpha2, scalar(0))
    *rho2/runTime.deltaT())
);
 surfaceScalarField alpharAUf1
(
  fvc::interpolate(max(alpha1, phase1.residualAlpha())*rAU1)
);
 surfaceScalarField alpharAUf2
(
  fvc::interpolate(max(alpha2, phase2.residualAlpha())*rAU2)
);

// --- Pressure corrector loop
while (pimple.correct())
{
 volVectorField HbyA1
(
  IOobject::groupName("HbyA", phase1.name()),
  U1
);
 HbyA1 =
  rAU1*(U1Eqn.H()+ max(phase1.residualAlpha() - alpha1, scalar(0))
   *rho1*U1.oldTime()/runTime.deltaT());

volVectorField HbyA2
```

```
(
  IOobject::groupName("HbyA", phase2.name()),
  U2
);
 HbyA2 =
  rAU2*(U2Eqn.H()+  max(phase2.residualAlpha() - alpha2, scalar(0))
   *rho2*U2.oldTime()/runTime.deltaT());

// Phase-1 predicted flux
surfaceScalarField phiHbyA1
(
  IOobject::groupName("phiHbyA", phase1.name()),
  fvc::flux(HbyA1)
  + phiCorrCoeff1*fvc::interpolate(alpha1.oldTime()*rho1.oldTime()*rAU1)
  *(MRF.absolute(phi1.oldTime())- fvc::flux(U1.oldTime()))/runTime.deltaT()
  - phiF1()
  - phig1
);

// Phase-2 predicted flux
surfaceScalarField phiHbyA2
(
  IOobject::groupName("phiHbyA", phase2.name()),
  fvc::flux(HbyA2)
  + phiCorrCoeff2*fvc::interpolate(alpha2.oldTime()*rho2.oldTime()*rAU2)
  *(MRF.absolute(phi2.oldTime())- fvc::flux(U2.oldTime()))/runTime.deltaT()
  - phiF2()
  - phig2
);

// Face-drag coefficients
surfaceScalarField rAUKd1(fvc::interpolate(rAU1*Kd));
surfaceScalarField rAUKd2(fvc::interpolate(rAU2*Kd));

// Construct the mean predicted flux including explicit drag contributions
surfaceScalarField phiHbyA
(
  "phiHbyA",
  alphaf1*(phiHbyA1 + rAUKd1*MRF.absolute(phi2))
  + alphaf2*(phiHbyA2 + rAUKd2*MRF.absolute(phi1))
);
MRF.makeRelative(phiHbyA);

surfaceScalarField rAUf
(
  "rAUf",
   mag(alphaf1*alpharAUf1 + alphaf2*alpharAUf2)
);
```

```
// Update the fixedFluxPressure BCs to ensure flux consistency
setSnGrad<fixedFluxPressureFvPatchScalarField>
(p_rgh.boundaryFieldRef(),
(phiHbyA.boundaryField()
- (alphaf1.boundaryField()*phi1.boundaryField()
 + alphaf2.boundaryField()*phi2.boundaryField())
 )/(mesh.magSf().boundaryField()*rAUf.boundaryField()));

// Iterate over the pressure equation to correct for non-orthogonality
while (pimple.correctNonOrthogonal())
{
 fvScalarMatrix pEqnIncomp
 (fvc::div(phiHbyA)- fvm::laplacian(rAUf, p_rgh));

solve
( pEqnComp1() + pEqnComp2() + pEqnIncomp,
  mesh.solver(p_rgh.select(pimple.finalInnerIter())));

// Correct fluxes and velocities on last non-orthogonal iteration
if (pimple.finalNonOrthogonalIter())
{
  phi = phiHbyA + pEqnIncomp.flux();
  surfaceScalarField mSfGradp("mSfGradp", pEqnIncomp.flux()/rAUf);

// Partial-elimination phase-flux corrector
{
  surfaceScalarField phi1s
  ( phiHbyA1 + alpharAUf1*mSfGradp );

  surfaceScalarField phi2s
  ( phiHbyA2 + alpharAUf2*mSfGradp);

 surfaceScalarField phir
  (((phi1s + rAUKd1*phi2s) - (phi2s + rAUKd2*phi1s))/(1 - rAUKd1*rAUKd2));
  phi1 = phi + alphaf2*phir;
  phi2 = phi - alphaf1*phir;
}

// Partial-elimination phase-velocity corrector
{
  volVectorField Us1
  (HbyA1+ fvc::reconstruct(alpharAUf1*mSfGradp - phiF1() - phig1));
  volVectorField Us2
  (HbyA2+ fvc::reconstruct(alpharAUf2*mSfGradp - phiF2() - phig2));

volScalarField D1(rAU1*Kd);
volScalarField D2(rAU2*Kd);

U = alpha1*(Us1 + D1*U2) + alpha2*(Us2 + D2*U1);
```

```
volVectorField Ur(((1 - D2)*Us1 - (1 - D1)*Us2)/(1 - D1*D2));

U1 = U + alpha2*Ur;
U1.correctBoundaryConditions();
fvOptions.correct(U1);

U2 = U - alpha1*Ur;
U2.correctBoundaryConditions();
fvOptions.correct(U2);

U = fluid.U();
   }
  }
}
// ************************************************************************* //
```

Appendice B

MultiphaseEulerFoam

Tutte le appendici che seguiranno, sono degli estratti del codice `multiphaseEulerFoam` della versione 5 di OpenFOAM. Le linee di codice complete sono reperibili sul sito https://openfoam.org. I titoli delle appendici sono i nomi dei file principali che compongono il solver `multiphaseEulerFoam`.

B.1 MultiphaseSystem.C

```
// * * * * * * * * * * Private Member Functions  * * * * * * * * * * * //
void Foam::multiphaseSystem::calcAlphas()
{ scalar level = 0.0;
   alphas_ == 0.0;

  forAllIter(PtrDictionary<phaseModel>, phases_, iter)
  { alphas_ += level*iter();
     level += 1.0;
  }
}

void Foam::multiphaseSystem::solveAlphas()
{ PtrList<surfaceScalarField> alphaPhiCorrs(phases_.size());
   int phasei = 0;

   forAllIter(PtrDictionary<phaseModel>, phases_, iter)
   {
    phaseModel& phase = iter();
    volScalarField& alpha1 = phase;

  alphaPhiCorrs.set
   (
    phasei,
    new surfaceScalarField
    ("phi" + alpha1.name() + "Corr",
      fvc::flux
      (phi_,
       phase,
      "div(phi," + alpha1.name() + ')')
```

```
    )
   );
surfaceScalarField& alphaPhiCorr = alphaPhiCorrs[phasei];

  forAllIter(PtrDictionary<phaseModel>, phases_, iter2)
   {
     phaseModel& phase2 = iter2();
     volScalarField& alpha2 = phase2;

     if (&phase2 == &phase) continue;
     surfaceScalarField phir(phase.phi() - phase2.phi());

     scalarCoeffSymmTable::const_iterator cAlpha
     (cAlphas_.find(interfacePair(phase, phase2)));

     if (cAlpha != cAlphas_.end())
      {
        surfaceScalarField phic
        ((mag(phi_) + mag(phir))/mesh_.magSf());

      phir += min(cAlpha()*phic, max(phic))*nHatf(phase, phase2);
      }

   word phirScheme
    ("div(phir," + alpha2.name() + ',' + alpha1.name() + ')');

   alphaPhiCorr += fvc::flux
       (
       -fvc::flux(-phir, phase2, phirScheme),
        phase,
        phirScheme
       );
   }
phase.correctInflowOutflow(alphaPhiCorr);

  MULES::limit
   (
     1.0/mesh_.time().deltaT().value(),
     geometricOneField(),
     phase,
     phi_,
     alphaPhiCorr,
     zeroField(),
     zeroField(),
     1,
     0,
     true
   );
  phasei++;
```

```
}
 MULES::limitSum(alphaPhiCorrs);

  volScalarField sumAlpha
  (
    IOobject
    ("sumAlpha",
      mesh_.time().timeName(),
      mesh_),
     mesh_,
    dimensionedScalar("sumAlpha", dimless, 0)
  );
 phasei = 0;

  forAllIter(PtrDictionary<phaseModel>, phases_, iter)
  {
    phaseModel& phase = iter();

  surfaceScalarField& alphaPhi = alphaPhiCorrs[phasei];
   alphaPhi += upwind<scalar>(mesh_, phi_).flux(phase);
   phase.correctInflowOutflow(alphaPhi);

   MULES::explicitSolve
  (
    geometricOneField(),
    phase,
    alphaPhi,
    zeroField(),
    zeroField()
  );
  phase.alphaPhi() = alphaPhi;

  Info<< phase.name() << " volume fraction, min, max = "
  << phase.weightedAverage(mesh_.V()).value()
  << ' ' << min(phase).value()
  << ' ' << max(phase).value() << endl;
  sumAlpha += phase;
  phasei++;
}
// * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * * //

Foam::autoPtr<Foam::multiphaseSystem::dragCoeffFields>
Foam::multiphaseSystem::dragCoeffs() const
{ autoPtr<dragCoeffFields> dragCoeffsPtr(new dragCoeffFields);
   forAllConstIter(dragModelTable, dragModels_, iter)
   { const dragModel& dm = *iter();

   volScalarField* Kptr =
   (
```

```
    max(dm.phase1()*dm.phase2(),
         dm.residualPhaseFraction())*dm.K
         (max(mag(dm.phase1().U() - dm.phase2().U()),
         dm.residualSlip()))
    ).ptr();
   }
  return dragCoeffsPtr;
}

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

B.2 UEqns.H

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

UEqns.set

  (
    phasei,
    new fvVectorMatrix
    ( fvm::ddt(alpha, U)+ fvm::div(phase.alphaPhi(), U)
     + (alpha/phase.rho())*fluid.Cvm(phase)*
       ( fvm::ddt(U) + fvm::div(phase.phi(), U)
        - fvm::Sp(fvc::div(phase.phi()), U)
       )
     - fvm::laplacian(alpha*nuEff, U)
     - fvc::div(alpha*(nuEff*dev(T(fvc::grad(U)))),"div(Rc)")
    ==
    (alpha/phase.rho())*fluid.Svm(phase)
     - fvm::Sp(slamDampCoeff*max(mag(U()) - maxSlamVelocity,
       dimensionedScalar("U0", dimVelocity, 0))
       /pow(mesh.V(), 1.0/3.0), U))
  );

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

B.3 SchillerNaumann.C

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

Foam::tmp<Foam::volScalarField> Foam::dragModels::SchillerNaumann::K
(const volScalarField& Ur)
 const
 {
  volScalarField Re(max(Ur*phase1_.d()/phase2_.nu(), scalar(1.0e-3)));
  volScalarField Cds
  (
   neg(Re - 1000)*(24.0*(1.0 + 0.15*pow(Re, 0.687))/Re)
  + pos0(Re - 1000)*0.44
  );
 return 0.75*Cds*phase2_.rho()*Ur/phase1_.d();
}

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

B.4 PEqn.H

```
 // ************************************************************************* //
    {  rAUs.set(phasei, (1.0/(UEqns[phasei].A() +  dragCoeffi  )).ptr());
       rAlphaAUfs.set
       (
         phasei,
         ( alphafs[phasei]/fvc::interpolate(UEqns[phasei].A()
          + dragCoeffi )).ptr()
       );
       HbyAs[phasei] = rAUs[phasei]*UEqns[phasei].H();

       phiHbyAs[phasei] =
       (fvc::flux(HbyAs[phasei])
        + rAlphaAUfs[phasei]*fvc::ddtCorr(phase.U(), phase.phi()));

       phiHbyAs[phasei] +=
       rAlphaAUfs[phasei]
       *(fluid.surfaceTension(phase)*mesh.magSf()
```

```
    + (phase.rho() - fvc::interpolate(rho))*(g & mesh.Sf())
    - ghSnGradRho)/phase.rho();

  multiphaseSystem::dragModelTable::const_iterator dmIter =
  fluid.dragModels().begin();
  multiphaseSystem::dragCoeffFields::const_iterator dcIter =
  dragCoeffs().begin();

  phiHbyAs[phasei] +=
  fvc::interpolate((*dcIter())/phase.rho())
  /fvc::interpolate(UEqns[phasei].A() + dragCoeffi)
  *phase2Ptr->phi();

  HbyAs[phasei] +=
  (1.0/phase.rho())*rAUs[phasei]*(*dcIter())*phase2Ptr->U();
  }
  phiHbyA += alphafs[phasei]*phiHbyAs[phasei];

  phasei++;
}

  phasei = 0;
  forAllIter(PtrDictionary<phaseModel>, fluid.phases(), iter)
  {
    phaseModel& phase = iter();
    rAUf += mag(alphafs[phasei]*rAlphaAUfs[phasei])/phase.rho();
    phasei++;
  }

  // Update the fixedFluxPressure BCs to ensure flux consistency
{
    surfaceScalarField::Boundary phib(phi.boundaryField());
    phib = 0;
    phasei = 0;
    forAllIter(PtrDictionary<phaseModel>, fluid.phases(), iter)
    {
     phaseModel& phase = iter();

     phib +=
      alphafs[phasei].boundaryField()
     *(mesh.Sf().boundaryField() & phase.U().boundaryField());

     phasei++;
  }

  setSnGrad<fixedFluxPressureFvPatchScalarField>
  (
   p_rgh.boundaryFieldRef(),
    (phiHbyA.boundaryField() - MRF.relative(phib)
```

```
    )/(mesh.magSf().boundaryField()*rAUf.boundaryField())
  );
 }

  while (pimple.correctNonOrthogonal())
  {
  fvScalarMatrix pEqnIncomp
  (fvc::div(phiHbyA)- fvm::laplacian(rAUf, p_rgh));

  pEqnIncomp.setReference(pRefCell, pRefValue);

  solve
  (pEqnIncomp,
  mesh.solver(p_rgh.select(pimple.finalInnerIter())));

  if (pimple.finalNonOrthogonalIter())
  {
  surfaceScalarField mSfGradp("mSfGradp", pEqnIncomp.flux()/rAUf);

  phasei = 0;
  phi = dimensionedScalar("phi", phi.dimensions(), 0);
  forAllIter(PtrDictionary<phaseModel>, fluid.phases(), iter)
   {
    phaseModel& phase = iter();

    phase.phi() =
     phiHbyAs[phasei]+ rAlphaAUfs[phasei]*mSfGradp/phase.rho();
    phi +=
     alphafs[phasei]*phiHbyAs[phasei]
     + mag(alphafs[phasei]*rAlphaAUfs[phasei])*mSfGradp/phase.rho();

    phasei++;
   }

  p_rgh.relax();

  p = p_rgh + rho*gh;

  mSfGradp = pEqnIncomp.flux()/rAUf;

  phasei = 0;
  forAllIter(PtrDictionary<phaseModel>, fluid.phases(), iter)
   {
    phaseModel& phase = iter();
    const volScalarField& alpha = phase;

    phase.U() =
     HbyAs[phasei] + fvc::reconstruct(
       rAlphaAUfs[phasei]*((phase.rho() - fvc::interpolate(rho))
```

```
            *(g & mesh.Sf())- ghSnGradRho+ mSfGradp))/phase.rho();

        phase.U().correctBoundaryConditions();

        U += alpha*phase.U();
        phasei++;
      }
    }
  }
// ************************************************************************* //
```

Appendice C

Modifiche al codice twoPhaseEulerFoam

Di seguito vengono mostrate le linee di codice relative alle modifiche effettuate nel solver twoPhaseEulerFoam.

C.1 dragModel.C

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

 Foam::tmp<Foam::volScalarField> Foam::dragModel::Ki() const
 {
   return
   0.75
   *CdRe()
   *swarmCorrection_->Cs()
   *pair_.continuous().rho()
   *pair_.magUr()
   /pair_.dispersed().d();
  }
 Foam::tmp<Foam::volScalarField> Foam::dragModel::K() const
  {
   return max(pair_.dispersed()* pair_.continuous(),
   pair_.dispersed().residualAlpha())*Ki();
  }

  Foam::tmp<Foam::surfaceScalarField> Foam::dragModel::Kf() const
 {
   return
   max(fvc::interpolate(pair_.dispersed()*pair_.continuous()),
    pair_.dispersed().residualAlpha())*fvc::interpolate(Ki());
 }
// ************************************************************************* //
```

C.2 SchillerNaumann.C

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

   Foam::tmp<Foam::volScalarField>
   Foam::dragModels::SchillerNaumann::CdRe() const
{
  volScalarField Re(max(pair_.Re(),residualRe_));
  return
  neg(Re - 1000)*24.0/Re*(1.0 + 0.15*pow(Re, 0.687))
  + pos0(Re - 1000)*0.44;
}
// ************************************************************************* //
```

Appendice D

Modifiche al codice `multiphaseEulerFoam`

Di seguito vengono mostrate le linee di codice relative alle modifiche effettuate nel solver `multiphaseEulerFoam`.

D.1 `multiphaseSystem.C` per 1 fase dispersa

```
// * * * * * * * * * * Private Member Functions  * * * * * * * * * * * //

 void Foam::multiphaseSystem::calcAlphas()
 {
  scalar level = 0.0;
  alphas_ == 0.0;

  forAllIter(PtrDictionary<phaseModel>, phases_, iter)
  {
   alphas_ += level*iter();
   level += 1.0;
  }
 }

  void Foam::multiphaseSystem::solveAlphas()
  {
  const dictionary& alphaControls = mesh_.solverDict("alpha");
  label nAlphaSubCycles(readLabel(alphaControls.lookup("nAlphaSubCycles")));
  label nAlphaCorr(readLabel(alphaControls.lookup("nAlphaCorr")));

  phaseModel& phase1 = phases_["air"];
  phaseModel& phase2 = phases_["water"];

  volScalarField& alpha1 = phase1;
  volScalarField& alpha2 = phase2;

  const surfaceScalarField& phi1 = phase1.phi();
  const surfaceScalarField& phi2 = phase2.phi();
```

```
 word alphaScheme("div(phi," + alpha1.name() + ')');
 word alpharScheme("div(phir," + alpha1.name() + ')');

 alpha1.correctBoundaryConditions();

 surfaceScalarField phic("phic", phi_);
 surfaceScalarField phir("phir", phi1 - phi2);

 for (int acorr=0; acorr<nAlphaCorr; acorr++)
{
  surfaceScalarField alphaPhic1
  (
   fvc::flux
   (phic,
    alpha1,
    alphaScheme
   )
  + fvc::flux
  (-fvc::flux(-phir, scalar(1) - alpha1, alpharScheme),
   alpha1,
   alpharScheme)
 );

 phase1.correctInflowOutflow(alphaPhic1);

 if (nAlphaSubCycles > 1)
 {
   for(subCycle<volScalarField> alphaSubCycle(alpha1, nAlphaSubCycles);
   !(++alphaSubCycle).end();)
    {
    surfaceScalarField alphaPhic10(alphaPhic1);

    MULES::explicitSolve
    (
     geometricOneField(),
     alpha1,
     phi_,
     alphaPhic10,
     zeroField(),
     zeroField(),
     1,
     0
    );

    if (alphaSubCycle.index() == 1)
     {phase1.alphaPhi() = alphaPhic10;}
    else
     {phase1.alphaPhi() += alphaPhic10;}
   }
```

```
    phase1.alphaPhi() /= nAlphaSubCycles;
   }
  else
  {
   MULES::explicitSolve
    (
     geometricOneField(),
     alpha1,
     phi_,
     alphaPhic1,
     zeroField(),//Sp,
     zeroField(),//Su,
     1,
     0
    );
   phase1.alphaPhi() = alphaPhic1;
   }

  phase2.alphaPhi() = phi_ - phase1.alphaPhi();
  phase2.correctInflowOutflow(phase2.alphaPhi());

  Info<< alpha1.name() << " volume fraction = "
  << alpha1.weightedAverage(mesh_.V()).value()
  << "  Min(" << alpha1.name() << ") = " << min(alpha1).value()
  << "  Max(" << alpha1.name() << ") = " << max(alpha1).value()
  << endl;

  alpha1.max(0);
  alpha1.min(1);
  alpha2 = scalar(1) - alpha1;
  }
}
// ************************************************************************* //
```

D.2 multiphaseSystem.C per 2 fasi disperse

```
// * * * * * * * * * * Private Member Functions  * * * * * * * * * * * //

void Foam::multiphaseSystem::calcAlphas()
{
     scalar level = 0.0;
     alphas_ == 0.0;

  forAllIter(PtrDictionary<phaseModel>, phases_, iter)
     {
    alphas_ += level*iter();
     level += 1.0;
    }
}

void Foam::multiphaseSystem::solveAlphas()
{
const dictionary& alphaControls = mesh_.solverDict("alpha");
label nAlphaSubCycles(readLabel(alphaControls.lookup("nAlphaSubCycles")));
label nAlphaCorr(readLabel(alphaControls.lookup("nAlphaCorr")));

phaseModel& phase1 = phases_["air1"];
phaseModel& phase2 = phases_["air2"];
phaseModel& phase3 = phases_["water"];

volScalarField& alpha1 = phase1;
volScalarField& alpha2 = phase2;
volScalarField& alpha3 = phase3;

const surfaceScalarField& phi1 = phase1.phi();
const surfaceScalarField& phi2 = phase2.phi();
const surfaceScalarField& phi3 = phase3.phi();

word alphaScheme("div(phi," + alpha1.name() + ')');
word alpharScheme("div(phir," + alpha1.name() + ')');

alpha1.correctBoundaryConditions();
alpha2.correctBoundaryConditions();

surfaceScalarField phic("phic", phi_);
surfaceScalarField phir1("phir1", phi1 - phi3);
surfaceScalarField phir2("phir2", phi2 - phi3);
```

```
for (int acorr=0; acorr<nAlphaCorr; acorr++)
{
  surfaceScalarField alphaPhic1
      (
   fvc::flux
    (phic,
     alpha1,
     alphaScheme)
   + fvc::flux
    (-fvc::flux(-phir1, scalar(1) - alpha1, alpharScheme),
        alpha1,
    alpharScheme
    )
   + fvc::flux
    (
   -fvc::flux(phir2, alpha2, alpharScheme),
    alpha1,
    alpharScheme
    )
 );

phase1.correctInflowOutflow(alphaPhic1);

surfaceScalarField alphaPhic2
(
  fvc::flux
  (phic,
   alpha2,
   alphaScheme)
  + fvc::flux
  (-fvc::flux(-phir2, scalar(1) - alpha2, alpharScheme),
   alpha2,
   alpharScheme)
  + fvc::flux
  (-fvc::flux(phir1, alpha1, alpharScheme),
   alpha2,
   alpharScheme)
);

phase2.correctInflowOutflow(alphaPhic2);

if (nAlphaSubCycles > 1)
{
 for
 (subCycle<volScalarField> alphaSubCycle(alpha1, nAlphaSubCycles);
  !(++alphaSubCycle).end();)
 {
  surfaceScalarField alphaPhic10(alphaPhic1);
```

```
  MULES::explicitSolve
  (
   geometricOneField(),
       alpha1,
   phi_,
   alphaPhic10,
   zeroField(),
   zeroField(),
   1,
   0
  );

 if (alphaSubCycle.index() == 1)
  {phase1.alphaPhi() = alphaPhic10;}
 else
  {phase1.alphaPhi() += alphaPhic10;}
}

phase1.alphaPhi() /= nAlphaSubCycles;

for
(subCycle<volScalarField> alphaSubCycle(alpha2, nAlphaSubCycles);
!(++alphaSubCycle).end();)
{
  surfaceScalarField alphaPhic20(alphaPhic2);

     MULES::explicitSolve
   (
    geometricOneField(),
      alpha2,
    phi_,
    alphaPhic20,
    zeroField(),
    zeroField(),
    1,
    0
   );

     if (alphaSubCycle.index() == 1)
  {phase2.alphaPhi() = alphaPhic20;}
  else
  {phase2.alphaPhi() += alphaPhic20;}
}

phase2.alphaPhi() /= nAlphaSubCycles;
}
else
{
```

```
  MULES::explicitSolve
  (
  geometricOneField(),
  alpha1,
  phi_,
  alphaPhic1,
  zeroField(),
  zeroField(),
  1,
  0
  );

phase1.alphaPhi() = alphaPhic1;
MULES::explicitSolve
(
  geometricOneField(),
  alpha2,
  phi_,
  alphaPhic2,
  zeroField(),
  zeroField(),
  1,
  0
);

phase2.alphaPhi() = alphaPhic2;
}

phase3.alphaPhi() = phi_ - phase1.alphaPhi()-phase2.alphaPhi();
phase3.correctInflowOutflow(phase3.alphaPhi());

Info<< alpha1.name() << " volume fraction = "
<< alpha1.weightedAverage(mesh_.V()).value()
<< "  Min(" << alpha1.name() << ") = " << min(alpha1).value()
<< "  Max(" << alpha1.name() << ") = " << max(alpha1).value()
<< endl;

Info<< alpha2.name() << " volume fraction = "
<< alpha2.weightedAverage(mesh_.V()).value()
<< "  Min(" << alpha2.name() << ") = " << min(alpha2).value()
<< "  Max(" << alpha2.name() << ") = " << max(alpha2).value()
<< endl;

alpha1.max(0);
alpha1.min(1);

alpha2.max(0);
alpha2.min(1);
```

```
alpha3 = scalar(1) - alpha1 - alpha2;
 }
}
// ************************************************************************* //
```

D.3 multiphaseSystem.C per 3 fasi disperse

```
// * * * * * * * * * * Private Member Functions  * * * * * * * * * * * //

void Foam::multiphaseSystem::calcAlphas()
{ scalar level = 0.0;
   alphas_ == 0.0;

  forAllIter(PtrDictionary<phaseModel>, phases_, iter)
 {alphas_ += level*iter();
   level += 1.0;}
}

void Foam::multiphaseSystem::solveAlphas()
{
const dictionary& alphaControls = mesh_.solverDict("alpha");
label nAlphaSubCycles(readLabel(alphaControls.lookup("nAlphaSubCycles")));
label nAlphaCorr(readLabel(alphaControls.lookup("nAlphaCorr")));

phaseModel& phase1 = phases_["air1"];
phaseModel& phase2 = phases_["air2"];
phaseModel& phase3 = phases_["air3"];
phaseModel& phase4 = phases_["water"];

volScalarField& alpha1 = phase1;
volScalarField& alpha2 = phase2;
volScalarField& alpha3 = phase3;
volScalarField& alpha4 = phase4;

const surfaceScalarField& phi1 = phase1.phi();
const surfaceScalarField& phi2 = phase2.phi();
const surfaceScalarField& phi3 = phase3.phi();
const surfaceScalarField& phi4 = phase4.phi();

word alphaScheme("div(phi," + alpha1.name() + ')');
word alpharScheme("div(phir," + alpha1.name() + ')');

alpha1.correctBoundaryConditions();
alpha2.correctBoundaryConditions();
alpha3.correctBoundaryConditions();

surfaceScalarField phic("phic", phi_);
surfaceScalarField phir1("phir1", phi1 - phi4);
```

```
surfaceScalarField phir2("phir2", phi2 - phi4);
surfaceScalarField phir3("phir3", phi3 - phi4);

for (int acorr=0; acorr<nAlphaCorr; acorr++)
{
  surfaceScalarField alphaPhic1
  (
   fvc::flux
   (phic,
    alpha1,
    alphaScheme
   )
      + fvc::flux
   (
   -fvc::flux(-phir1, scalar(1) - alpha1, alpharScheme),
    alpha1,
    alpharScheme
   )
   + fvc::flux
   (
   -fvc::flux(phir2, alpha2, alpharScheme),
    alpha1,
    alpharScheme
   )
   + fvc::flux
   (
   -fvc::flux(phir3, alpha3, alpharScheme),
    alpha1,
    alpharScheme
   )
  );

phase1.correctInflowOutflow(alphaPhic1);

surfaceScalarField alphaPhic2
(
 fvc::flux
 (
      phic,
  alpha2,
  alphaScheme
 )
 + fvc::flux
 (-fvc::flux(-phir2, scalar(1) - alpha2, alpharScheme),
 alpha2,
 alpharScheme)
 + fvc::flux
 (-fvc::flux(phir1, alpha1, alpharScheme),
 alpha2,
```

```
 alpharScheme)
 + fvc::flux
 (-fvc::flux(phir3, alpha3, alpharScheme),
 alpha2,
 alpharScheme)
);

phase2.correctInflowOutflow(alphaPhic2);

surfaceScalarField alphaPhic3
(
 fvc::flux
 (
 phic,
 alpha3,
 alphaScheme
 )
 + fvc::flux
 (-fvc::flux(-phir3, scalar(1) - alpha3, alpharScheme),
  alpha3,
  alpharScheme)
 + fvc::flux
 (-fvc::flux(phir1, alpha1, alpharScheme),
  alpha3,
  alpharScheme)
 + fvc::flux
 (-fvc::flux(phir2, alpha2, alpharScheme),
  alpha3,
  alpharScheme)
);

phase3.correctInflowOutflow(alphaPhic3);

if (nAlphaSubCycles > 1)
  {
for
(subCycle<volScalarField> alphaSubCycle(alpha1, nAlphaSubCycles);
!(++alphaSubCycle).end();)
{
surfaceScalarField alphaPhic10(alphaPhic1);

MULES::explicitSolve
(
  geometricOneField(),
  alpha1,
  phi_,
  alphaPhic10,
  zeroField(),
  zeroField(),
```

```
  1,
  0
);

if (alphaSubCycle.index() == 1)
{phase1.alphaPhi() = alphaPhic10;}
else
{phase1.alphaPhi() += alphaPhic10;}
}

phase1.alphaPhi() /= nAlphaSubCycles;

for
(subCycle<volScalarField> alphaSubCycle(alpha2, nAlphaSubCycles);
!(++alphaSubCycle).end();)
{
  surfaceScalarField alphaPhic20(alphaPhic2);

  MULES::explicitSolve
  (
   geometricOneField(),
   alpha2,
   phi_,
   alphaPhic20,
   zeroField(),
   zeroField(),
   1,
   0
  );

    if (alphaSubCycle.index() == 1)
 {phase2.alphaPhi() = alphaPhic20;}
 else
 {phase2.alphaPhi() += alphaPhic20;}
}

phase2.alphaPhi() /= nAlphaSubCycles;

for
(subCycle<volScalarField> alphaSubCycle(alpha3, nAlphaSubCycles);
!(++alphaSubCycle).end();)
{
  surfaceScalarField alphaPhic30(alphaPhic3);

MULES::explicitSolve
(
  geometricOneField(),
  alpha3,
  phi_,
```

```
  alphaPhic30,
  zeroField(),
  zeroField(),
  1,
  0
);

 if (alphaSubCycle.index() == 1)
 {phase3.alphaPhi() = alphaPhic30;}
 else
 {phase3.alphaPhi() += alphaPhic30;}
}

phase3.alphaPhi() /= nAlphaSubCycles;

  }
   else
{
MULES::explicitSolve
(
  geometricOneField(),
  alpha1,
  phi_,
  alphaPhic1,
  zeroField(),
  zeroField(),
  1,
  0
);

phase1.alphaPhi() = alphaPhic1;

MULES::explicitSolve
(
  geometricOneField(),
  alpha2,
  phi_,
  alphaPhic2,
  zeroField(),
  zeroField(),
  1,
  0
);

phase2.alphaPhi() = alphaPhic2;

MULES::explicitSolve
(
  geometricOneField(),
```

```
  alpha3,
  phi_,
  alphaPhic3,
  zeroField(),
  zeroField(),
  1,
  0
);
phase3.alphaPhi() = alphaPhic3;
}

phase4.alphaPhi() = phi_ - phase1.alphaPhi() +
                    -phase2.alphaPhi()-phase3.alphaPhi();
phase4.correctInflowOutflow(phase4.alphaPhi());

Info<< alpha1.name() << " volume fraction = "
<< alpha1.weightedAverage(mesh_.V()).value()
<< "  Min(" << alpha1.name() << ") = " << min(alpha1).value()
<< "  Max(" << alpha1.name() << ") = " << max(alpha1).value()
<< endl;

Info<< alpha2.name() << " volume fraction = "
<< alpha2.weightedAverage(mesh_.V()).value()
<< "  Min(" << alpha2.name() << ") = " << min(alpha2).value()
<< "  Max(" << alpha2.name() << ") = " << max(alpha2).value()
<< endl;

Info<< alpha3.name() << " volume fraction = "
<< alpha3.weightedAverage(mesh_.V()).value()
<< "  Min(" << alpha3.name() << ") = " << min(alpha3).value()
<< "  Max(" << alpha3.name() << ") = " << max(alpha3).value()
<< endl;

alpha1.max(0);
alpha1.min(1);

alpha2.max(0);
alpha2.min(1);

alpha3.max(0);
alpha3.min(1);

alpha4 = scalar(1) - alpha1 - alpha2 - alpha3;
}
}
// ************************************************************************* //
```

D.4 pEqn.H

```
// ************************************************************************* //
 rAUs.set(phasei, (1.0/(UEqns[phasei].A() + dragCoeffi
 + (max(phase.residualAlpha() - alpha,
      scalar(0))/runTime.deltaT()))).ptr());

 rAlphaAUfs.set
  (
   phasei,
   (fvc::interpolate(max(alpha, phase.residualAlpha())*(
     1.0/(UEqns[phasei].A()+ dragCoeffi + (max(phase.residualAlpha()
         - alpha, scalar(0))/runTime.deltaT()))))).ptr())
 );

  HbyAs[phasei] = rAUs[phasei]*(UEqns[phasei].H()
      + max(phase.residualAlpha()
      - alpha, scalar(0))*U.oldTime()/runTime.deltaT());

  phiHbyAs[phasei] =
   (
     fvc::flux(HbyAs[phasei])
    + rAlphaAUfs[phasei]*fvc::ddtCorr(phase.U(), phase.phi())
   );

  phiHbyAs[phasei] +=
  rAlphaAUfs[phasei]*(
   fluid.surfaceTension(phase)*mesh.magSf()
   + (phase.rho() - fvc::interpolate(rho))*(g & mesh.Sf())- ghSnGradRho
      )/phase.rho();

  phiHbyAs[phasei] +=
   fvc::interpolate((*dcIter())/phase.rho())
   /fvc::interpolate(UEqns[phasei].A() + dragCoeffi
  + (max(phase.residualAlpha() - alpha, scalar(0))/runTime.deltaT()))
   *phase2Ptr->phi();

  HbyAs[phasei] +=
  (1.0/phase.rho())*rAUs[phasei]*(*dcIter())*phase2Ptr->U();

  phiHbyA += alphafs[phasei]*phiHbyAs[phasei];

  phasei++;
```

```
    }

    surfaceScalarField rAUf
    (
     IOobject
      (
        "rAUf",
        runTime.timeName(),
        mesh
      ),
       mesh,
       dimensionedScalar("rAUf", dimensionSet(-1, 3, 1, 0, 0), 0)
    );

    phasei = 0;
    forAllIter(PtrDictionary<phaseModel>, fluid.phases(), iter)
    {
        phaseModel& phase = iter();
        rAUf += mag(alphafs[phasei]*rAlphaAUfs[phasei])/phase.rho();

        phasei++;
     }

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

Ringraziamenti

Vorrei ringraziare il prof. Vanni e il prof. Buffo, oltre che per la conoscenza fornitami, per la disponibilità e la precisione dimostratemi durante tutto il periodo di stesura.
Un grande ringraziamento a mia madre e mio padre che, con il loro instancabile sostegno, sia morale che economico, mi hanno permesso di arrivare fin qui, lasciandomi piena libertà di scelta su ogni aspetto.
Infine un grazie a tutti gli amici che nonostante la lontananza sono stati sempre presenti, e a coloro che hanno avuto un peso determinante nel conseguimento di questo risultato, punto di arrivo e contemporaneamente di partenza della mia vita.

Bibliografia

[1] Almstedt A. E. and Van Wachem B. G. M. 2003, Methods for multiphase computational fluid dynamics. *Chemical Engineering Journal*, 96 (1): 81 – 98.

[2] Andersson B. et al. *2011, Computational fluid dynamics for engineers*. Cambridge University Press, Cambridge.

[3] Becker S. and Pfleger D. 2001, Modelling and simulation of the dynamic flow behaviour in a bubble column. *Chemical Engineering Science*, 56 (4): 1737 – 1747.

[4] Bhusare V. H., Dhiman M. K., et al. 2017, CFD simulations of a bubble column with and without internals by using openfoam. *Chemical Engineering Journal*, 317 : 157–174.

[5] Buffo A., Marchisio D. L, Vanni M., and Renze P. 2013, Simulation of polydisperse multiphase systems using population balances and example application to bubbly flows. *Chemical Engineering Research and Design*, 91 (10): 1859 – 1875.

[6] Buwa V. V. and Ranade V. V. 2002, Dynamics of gas-liquid flow in a rectangular bubble column: experiments and single/multi-group cfd simulations. *Chemical Engineering Science*, 57 (22-23): 4715–4736.

[7] Gentric C. 2008, CFD simulation of the flow field in a bubble column reactor: Importance of the drag force formulation to describe regime transitions. *Chemical Engineering and Processing: Process Intensification*, 47 (9): 1726 – 1737.

[8] Carneiro J. N. E., Kaufmann V., and Polifke W. 2008, Implementation of a moments model in openfoam for polydispersed multiphase flows. In *Open Source CFD International Conference, Garching, Germany*.

[9] Diaz M. E., Iranzo A., Cuadra D., et al. 2008, Numerical simulation of the gas-liquid flow in a laboratory scale bubble column: influence of bubble size distribution and non-drag forces. *Chemical Engineering Journal*, 139 (2): 363–379.

[10] Greenshields C. J. 2017, Openfoam user guide. *OpenFOAM Foundation Ltd, version 5*.

[11] Ishii M. *1975, Thermo-Fluid Dynamic Theory of Two-Phase Flow, Eyrolles, Paris*.

[12] Krishna R. and Van Baten J. M. 2001, Scaling up bubble column reactors with the aid of CFD. *Chemical Engineering Research and Design*, 79 (3): 283 – 309.

[13] Miller T. F. and Miller D. J. 2003, A fourier analysis of the IPSA/PEA algorithms applied to multiphase flows with mass transfer. *Computers and fluids*, 32 (2): 197–221.

[14] Mishima K. and Ishii M. 1984 Two-fluid model and hydrodynamic constitutive relations. *Nuclear Engineering and Design*, 82 (2): 107 – 126.

[15] Naumann Z. and Schiller L. 1935, A drag coefficient correlation. *Z. Ver Deutsch. Ing*, 77 : 318–323.

[16] Oliveira P. J. and Issa R. I. 1994, On the numerical treatment of interphase forces in two-phase flow. *Numerical Methods in Multlphase Flow*, 185 : 131–131.

[17] Passalacqua A. and Fox R. 2011, Implementation of an iterative solution procedure for multi-fluid gas–particle flow models on unstructured grids. *Powder technology*, 213 (1-3): 174–187.

[18] Pfleger D., Gomes S., Gilbert N., and Wagner H. G. 1999, Hydrodynamic simulations of laboratory scale bubble columns fundamental studies of the eulerian–eulerian modelling approach. *Chemical Engineering Science*, 54 (21): 5091–5099.

[19] Schwarzkopf J. D., Sommerfeld M., Crowe C. T., and Tsuji Y. *2011, Multiphase flows with droplets and particles*. CRC press, New York.

[20] Silva L. F. L. R. and Lage P. L. C. 2007, Implementation of an eulerian multiphase model in openfoam and its application to polydisperse two-phase flows. In *OpenFOAM international conference, Rio de Janeiro, Brazil*.

[21] Sokolichin A., Eigenberger, et al. 1997, Dynamic numerical simulation of gas-liquid two-phase flows euler-euler versus euler-lagrange. *Chemical engineering science*, 52 (4): 611–626.

[22] Spalding B. D. 1981, Numerical computation of multiphase fluid flow and heat transfer. *Contribution to Recent Advances in Numerical Methods in Fluids*, 1: 161–191.

[23] Venier C. M., Damian S. M., and Nigro N. M. 2016, Numerical aspects of eulerian gas–particles flow formulations. *Computers and Fluids*, 133 : 151–169.

[24] Webb C., Que F., and Senior P. R. 1992, Dynamic simulation of gas-liquid dispersion behaviour in a 2-D bubble column using a graphics mini-supercomputer. *Chemical engineering science*, 47 (13-14): 3305–3312.