Department of Environment, Land and Infrastructure Engineering

Master of Science in Petroleum Engineering

"MITIGATION OF SPURIOUS CURRENTS IN MULTIPHASE FLOW SIMULATION AT LOW CAPILLARY NUMBERS"

Supervisor:

Prof. Dario Viberti

Co-Supervisor:

Dott. Filippo Panini

Candidate:

ABDALLAH KAMEL MOHAMED AHMED ELHAMADY [S289668]

July 2022

Acknowledgment

“And whatever you have of favor - it is from ALLAH” The Qur’an, Surah An-Nahl, 53 (QS 16: 53)

All praises to ALLAH for all His blessings and for the strength He provided me throughout the whole journey to reach this point, completing my master's degree. My greatest appreciation and respect go to Prof. Dario Viberti, first for his guidance, his encouragement, his knowledge and inspiring me during the whole master program, and second for giving me the opportunity to do my thesis under his supervision. Many special thanks go to my co-supervisor Dott. Filippo Panini for his invaluable insights, continuous support, and patience. Also, I would like to thank my friends and colleagues for their help and motivation.

Finally, I would like to express my very warm gratitude to my parents and siblings for their infinite love and support, for whom I spare no effort to make them proud. Nothing would have ever happened without you.

Table of contents

Index of figures

List of abbreviations

CFD	**C**omputational **F**luid **D**ynamics
OpenFOAM	**O**pen **F**ield **O**peration **A**nd **M**anipulation
VOF	**V**olume **O**f **F**luid Method
CMI	**C**lay **M**athematics **I**nstitute
NSE	**N**avier-**S**tokes **E**quations
FVM	**F**inite **V**olume **M**ethod
FEM	**F**inite **E**lement **M**ethod
FDM	**F**inite **D**ifference **M**ethod
CSF	**C**ontinuum **S**urface **F**orce
GUI	**G**raphical **U**ser **I**nterface
DNS	**D**irect **N**umerical **S**imulations
LES	**L**arge **E**ddy **S**imulations
RANS	**R**eynolds **A**veraged **N**avier-**S**tokes
IC	**I**nitial **C**onditions
BC	**B**oundary **C**onditions

Nomenclature

t	time, s
ρ	fluid mass density, kg/m^3
μ	viscosity, Pa.s
$\vec{\boldsymbol{u}}$	fluid velocity vector, m/s
$\hat{\boldsymbol{n}}$	normal unit vector
Ω	control volume
$\partial\Omega$	control surface
Kn	Knudsen number, dimensionless
λ	mean free path of the fluid particles at the scale of interest, m
∇	gradient differential operator
g	gravitational acceleration, m/s^2
$\otimes$	dyadic product
P	hydrostatic pressure, Pa
σ	stress tensor [matrix]
σ_{ii}	normal stress, Pa
τ_{ij}	shear stress, Pa
U	characteristic flow velocity scale, m/s
L	characteristic flow length scale, m
R_e	Reynold Number, dimensionless
Q	flow rate, m^3/s
u_{max}	maximum velocity, m/s
$\overline{u}$	average velocity, m/s
α	VOF index function, fraction
$\tilde{\alpha}$	smoothed VOF index function, fraction
N_c	Capillary Number, dimensionless
u_d	Darcy velocity, m/s
κ	curvature of the interface
A	area, m^2

S	square side length, m
R	radius, m
$\mathbf{f_s}$	force term due to the interfacial tension, N
u_r	relative compression velocity, m/s
u_w	wetting fluid velocity, m/s
u_{nw}	nonwetting fluid velocity, m/s
n_f	normal vector to the cell face
$C_\propto$	an adjustable coefficient indicating the level of the interface compression
F	volume flux across the cell face, m^3/s
S_f	cell surface area, m^2
γ	surface tension, N/m
p_{exact}	exact solution of Laplace pressure, Pa
$p_{simulation}$	numerical solution of Laplace pressure, Pa

Abstract

One of the most challenging issues in computational modelling of a multiphase flow is spurious currents. Spurious currents arise by the errors in the interface curvature computations between two fluids in the computational domain. Being predominant in porous media at small capillary numbers, they might cause numerical instabilities and compromise the quality of the results in terms of fluid velocity and capillary pressure within the numerical simulation. In this work, as an attempt to fix this issue, we investigate and implement the solutions proposed in the literature to develop a set of optimal computational settings that mitigate their impact. First, we provide numerical simulations for the classical case of a circular stationary droplet of oil relaxing from an initial square shape surrounded by water using the open-source finite-volume computational fluid dynamics (CFD) code OpenFOAM. The investigation is conducted setting the interfacial compression coefficient (C_α) equal to zero (no interface compression) and higher than zero. Then, in order to reduce spurious currents, the volume of fluid (VOF) index function is smoothed by applying the so-called smoother 'Laplacian filter' aiming to decrease the curvature computation errors. It was proved that applying the Laplacian filter twice smooths sufficiently the index function. Since this filter is not implemented in the OpenFOAM library, an external code found in the literature is used and linked to the VOF solver. Furthermore, an error analysis on the accuracy of the numerical solution of the Laplace pressure for a stationary droplet with respect to the exact (analytical) solution of the Laplace pressure for a sphere has been performed. The obtained results show that introducing the interface compression has led to a sharp (thinner) interface, but increased the magnitude of spurious currents which has been reduced later by almost one order of magnitude when the smoother was applied. The error in the Laplace pressure calculations has also decreased by almost 36%. This work demonstrates the potential and stability of the proposed computational rules to adequately reduce the magnitude of spurious currents when simulating multiphase flow phenomena in real porous media.

Chapter 1: Introduction

Due to the relevance and complex nature of multiphase flows in porous media, numerical simulations, and particularly computational fluid dynamics (CFD), have rapidly arisen as a promising and powerful tool to understand multiphase flows (Vachaparambil et al., 2019) in most oil and gas applications, such as natural gas storage, geological carbon dioxide storage, enhanced oil recovery (EOR) (Raeini et al., 2012). One of the key parameters of concern in numerical simulations is modelling the interface between two immiscible fluids. To do so, CFD strategies can be broadly divided into two categories: Lagrangian and Eulerian (Vachaparambil et al., 2019). Both have been used with considerable success, each with its own advantages and disadvantages (Mei, 2001). Lagrangian methods (or Front-Tracking methods) consider each one of the fluid particles as a discrete phase, track them and describe the variations in physical quantities (such as density, velocity, pressure, temperature, stresses, … etc.) as functions of time around each individual particle along its own path. Therefore, they accurately resolve the interface shape and apply the interfacial boundary conditions at the exact position of the interface (Francois, 1998). However, it is a bit complicated to use them for problems with large interface movements and severe topological changes. On the other hand, Eulerian methods (or Front-Capturing methods, like volume-of-fluid (VOF) method and Level-Set (LS) method) are ideally suited to handle such complex problems (Hoang et al., 2013). Variations in the physical quantities are described, as a function of time, at fixed points by which different particles pass at different times. Eulerian methods, in contrast to Lagrangian ones, employ a fixed grid and the fluid interface is not explicitly tracked but reconstructed from appropriate field variables (Francois, 1998).

Despite its simplicity, VOF method is considered one of the most robust methods used in multiphase simulations (Hirt et al., 1979). Simple, because it is based on the concept of the volume of fluid index function (α) in each cell of the computational domain, where $0 \leq \alpha \leq 1$. A value of 0 indicates one phase and a value of 1 indicates the second phase. Values between 0 and 1 refer to the fluid interface, where both phases are present. It is such popular that it is implemented in many CFD packages: both commercial and open-source ones.

A well-known associated problem is spurious currents, which have been recently an active topic of research (Vachaparambil et al., 2019). Spurious currents lead to unphysical fluid dynamics which adversely affect the predictive capability of these simulations and compromise the quality of the obtained results, like fluid velocity and capillary pressure. Unfortunately, they are predominant in low capillary number flows which typically characterize the porous media problems of our interest. They are mainly induced due to the numerical errors in the surface tension calculations in Front-Capturing methods, such as volume-of-fluid (VOF) method. At the expense of significant numerical cost, special treatment procedures must be followed to reduce spurious currents and thus accurately model low capillary number flows. Two techniques can be used to treat this problem, one is using an extra different field variable (like a height-function or a level-set function) only for calculating the interface curvature. This should help accurately calculate the surface tension. The other is smoothing the VOF index function over the interface region. This would result in avoiding the sharp steep gradient of the VOF index function over this thin interface region, and hence improving the surface tension calculations. Consequently, spurious currents would significantly decrease.

One of the most commonly used methods to model surface tension forces is the Continuum Surface Force (CSF). CSF, which is implemented in openFOAM, represents the fluid interface as a transition region and thus alleviates these previous interface topology constraints without sacrificing accuracy.

This study is organized as follows. We start with a short description for the theories behind OpenFOAM. Then, we present a well-characterized benchmark (static droplet) as an attempt to find the optimal computational rules that can be extended to real porous media.

Chapter 2: Literature review

2.1. Navier-Stokes Equations

2.1.1. Overview

Despite the fact that the motion of fluids has always been of the utmost importance for human beings, the evolution of mathematical models only emerged after the industrial revolution at the end of 19th century. The first appropriate description of the viscous fluid motion had been proposed in "Principia" paper written by Sir Isaac Newton (1687), presenting the dynamic behavior of fluids under constant viscosity. Over time, Daniel Bernoulli (1738) and Leonhard Euler (1755) had provided an equation for the inviscid fluid flow, known as Euler's equation. These Euler's equations, which predate the Navier–Stokes equations by many decades, do not account for the viscosity effects. Then, Navier-Stokes equations were developed over several decades. Claude Louis Marie Henri Navier (1827), Augustin Louis Cauchy (1828), Siméon Denis Poisson (1829), and Adhémar St.Venant (1843) had all carried out independently their studies on Euler's equations introducing the viscous (frictional) force. In 1845, Sir George Gabriel Stokes had derived the motion equation for a viscous flow by adding Newtonian viscous terms. Finally, the Navier-Stokes Equations had been brought to their final form which has been used to provide numerical solutions for fluid flow ever since (White, 1991) (Stokes, 1851).

The Navier–Stokes equations are considered the heart of fluid flow modelling (Heywood, 2006). When solved for a certain set of boundary conditions, like walls, inlets and outlets, they can predict both the fluid pressure and velocity for a given geometry. They can be used to model ocean currents, the weather, water flow in a pipe, air flow around a plane wing and many other fluids accurately over a wide range of conditions. Since they provide a rigorous analysis for the physics of many phenomena of scientific and engineering interest, they can help with the design of aircraft and cars, the analysis of pollution, the blood flow studies, the power stations design, ...etc. Moreover, they can be used to model magneto-hydrodynamics if coupled with Maxwell's equations.

The Navier-Stokes equations mainly govern the motion of fluids and describe how the velocity, pressure, temperature, and density of a moving fluid are related. They are just extensions of the Euler Equations, but include the viscosity effects on the flow. They are a set of coupled partial differential equations (not ordinary ones) and theoretically could be solved for a given flow problem using calculus methods. For instance, it is relatively easy to solve these equations for a flow between two parallel plates or for the flow in a cylinder, as we shall see. But, in practice and due to their complex nonlinearity, these equations are too difficult to be solved analytically. The fundamental issues of uniqueness and smoothness of their physically reasonable solutions in three-dimensional cases actually are relatively challenging and still are open problems (Friedlander, 2006). Even, the Clay Mathematics Institute (CMI) of Cambridge in Massachusetts, U.S. has considered it as a “Millennium Problem”, one of seven mathematical problems selected by the institute, and has offered a USD 1-million award for its solution (Mclean, 2012) (claymath.org).

In the past, engineers used to make further approximations and simplifications to the equations till they had a set of equations that they could solve. Recently, super speed computers have been used to solve approximations to the equations using a set of different techniques like finite difference, finite element, finite volume, and spectral, referring to Computational Fluid Dynamics (CFD) (grc.nasa.gov/www/k-12/rocket/nseqs.html).

The Navier-Stokes equations are based on two assumptions. First, the fluid is a continuum, a continuous substance rather than discrete particles. Second, the quantities of interest like pressure, temperature, density and flow velocity are sufficiently smooth or, in other words, weakly differentiable. In such case, the representative physical length scale of the system is much larger than the mean free path of the fluid molecules. The ratio of the mean free path of the fluid particles at the scale of interest, λ, to the representative length scale of the system, L, is referred as the Knudsen number, $Kn = \lambda/L$. As long as $Kn < 0.01$, the NS equations have been found valid. For $0.01< Kn <0.1$, they may be still used, but require special boundary conditions. For $Kn > 0.1$, they become invalid anymore. For instance, at the ambient pressure of 1 atm, the mean free path of air molecules – is 68 nanometers. The characteristic length of your model should therefore be larger than 6.8 μm so that the NSE can be valid.

Furthermore, the Navier–Stokes equations have been recently used extensively in video games and computer animation industry in order to model a wide variety of natural phenomena, such as simulations of small-scale gaseous fluids (fire and smoke) (Stam, 2003).

Mathematically, the Navier–Stokes equations are expressed based on the principles of conservation of mass, conservation of momentum and conservation of energy. Sometimes, they are accompanied by an equation of state relating the pressure, temperature and density of the gas (McLean, 2012).

Conservation of Mass: Continuity Equation
Conservation of Momentum: Newton's Second Law
Conservation of Energy: first Law of thermodynamics (referred as Energy Equation)

There are various ways for deriving these equations. In this work, the classical one of continuum mechanics will be used. Since we consider an isothermal flow, a constant-temperature flow, the Navier-Stokes equations are reduced to represent only the first two conservation laws – the conservation of mass and the conservation of momentum.

Firstly, let us consider a finite arbitrary volume, called a control volume, over which these principles can be applied. This control volume is denoted by Ω and its bounding control surface is ∂Ω. Ω must be a reasonably large finite volume of the fluid.

2.1.2. Navier-Stokes equations derivation

2.1.2.1. Continuity equation

The mass continuity equation, or simply the continuity equation, states the preservation of mass. The mass in the control volume cannot neither be created nor destroyed, i.e. the mass flow difference through the system between inlet and outlet is zero.

To formulate this definition mathematically, we proceed as follows

$$\underbrace{\frac{\partial}{\partial t}\int_{\Omega}\rho dV}_{\substack{change\ of\ mass\\ in\ time}} + \underbrace{\oint_{\partial\Omega}\rho\vec{\boldsymbol{u}}\,\widehat{\boldsymbol{n}}dS}_{\substack{Flow\ of\ mass\\ through\ the\\ boundaries}} = \underbrace{0}_{\substack{mass\ is\ not\ created\\ or\ destroyed}} \qquad \text{Eq. (1)}$$

where

t is time

ρ is the fluid mass density

$\vec{\boldsymbol{u}}$ is the fluid velocity vector

$\hat{\boldsymbol{n}}$ is the normal unit vector

Since the boundaries are fixed and do not change in time:

$$\frac{\partial}{\partial t}\int_{\Omega}\rho dV = \int_{\Omega}\frac{\partial}{\partial t}\rho dV = \int_{\Omega}\frac{\partial\rho}{\partial t}dV \qquad \text{Eq. (2)}$$

The surface integral is transcribed into a volume integral using divergence theorem (Gasuss' theorem), based on the concept that all the mass that *diverges* out of the control volume must inherently pass by the boundary of the control volume at a certain time:

$$\oint_{\partial\Omega}\rho\vec{u}\,\hat{n}dS = \int_{\Omega}\nabla.(\rho\vec{\boldsymbol{u}})\,dV \qquad \text{Eq. (3)}$$

where

∇ indicates the gradient differential operator

Substituting from Eq. (2) and (3) into Eq. (1):

$$\int_{\Omega}\frac{\partial\rho}{\partial t}dV + \int_{\Omega}\nabla.(\rho\vec{\boldsymbol{u}})\,dV = 0$$

By integration:

$$\frac{\partial\rho}{\partial t} + \nabla.(\rho\vec{\boldsymbol{u}}) = 0 \qquad \text{Eq (4)}$$

For an incompressible fluid, i.e. ρ is constant, Eq (4) becomes

$$\nabla.\vec{\boldsymbol{u}} = \boldsymbol{0} \qquad \text{Eq (5)}$$

2.1.2.2. Momentum conservation

The conservation of momentum implies that the momentum in a control volume is kept constant. According to Newton's second law, mass times acceleration is equal to the sum of forces that act on a volume unit.

The total force **F** acting on the control volume consists of two parts: surface and volume (body) forces:

$$\sum F = F_{surface} + F_{volume}$$

The surface forces are formally expressed in terms of the stress tensor matrix:

$$\underbrace{\sigma}_{Stress\ tensor} = \begin{pmatrix} \sigma_{xx} & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_{yy} & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_{zz} \end{pmatrix}$$

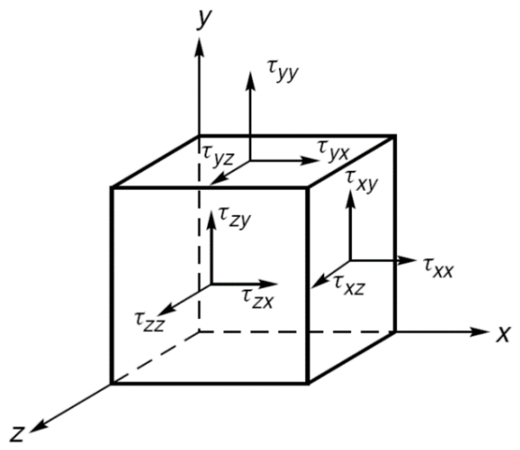


Fig. 1. Stress components on an infinitesimal rectangular (Salih, 2012)

There are two contributions to this stress tensor matrix:

- a contribution due to normal forces resulting from pressure gradients (ex. pressure), referring to the diagonal components of the stress tensor
- a contribution due to shear forces resulting from gradients in the fluid flow velocity (ex. viscous shear), referring to the non-diagonal components of the stress tensor.

The volume forces are those forces exerted on the fluid by external fields like gravity.

This is expressed mathematically as follows

$$\frac{\partial}{\partial t}\int_{\Omega}\rho\vec{\boldsymbol{u}}dV + \underbrace{\oint_{\partial\Omega}\rho\,(\vec{\boldsymbol{u}}\otimes\vec{\boldsymbol{u}}).\hat{n}dS}_{\text{Nonlinear advection term}} = \underbrace{\oint_{\partial\Omega}\boldsymbol{\sigma}.\hat{\boldsymbol{n}}dS}_{\substack{\text{Surface forces}\\ \text{(Normal and Shear)}}} + \underbrace{\int_{\Omega}\rho g dV}_{\text{Volume (Body)forces}} \qquad \text{Eq. (6)}$$

where

g is the gravitational acceleration

$\otimes$ represents the dyadic product

The stress tensor matrix is split up into two terms:

$$\underbrace{\sigma}_{Stress\ tensor} = \underbrace{\sigma^I}_{Isotropic\ term} + \underbrace{\sigma^D}_{Deviatoric\ term}$$

$$\underbrace{\sigma}_{Stress\ tensor} = -\begin{pmatrix} P & 0 & 0 \\ 0 & P & 0 \\ 0 & 0 & P \end{pmatrix} + \begin{pmatrix} P+\sigma_{xx} & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & P+\sigma_{yy} & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & P+\sigma_{zz} \end{pmatrix}$$

where P is the hydrostatic pressure: $P = -\frac{1}{3}\,(\sigma_{xx}+\sigma_{yy}+\sigma_{zz})$

Considering a Newtonian fluid where the stress tensor is assumed to be linearly related to the rate-of-strain tensor. For example, in the 1-D case: $\tau_{yx} = \mu \frac{\partial u_x}{\partial y}$

In the general case:

$$\tau_{ij} = \mu \left(\frac{\partial u_i}{\partial j} + \frac{\partial u_j}{\partial i}\right)$$

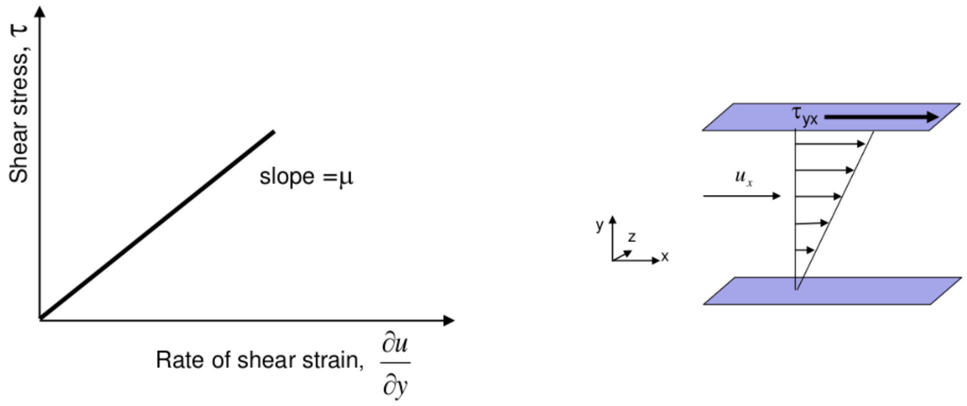


Fig. 2. Shear Stress for Newtonian fluids

Thus, the stress tensor can be reformulated as follows:

$$\sigma = -\begin{pmatrix} P & 0 & 0 \\ 0 & P & 0 \\ 0 & 0 & P \end{pmatrix} + \left\{ \begin{pmatrix} \sigma_{xx} & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_{yy} & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_{zz} \end{pmatrix} + \begin{pmatrix} -\frac{1}{3}\left(\sigma_{xx}+\sigma_{yy}+\sigma_{zz}\right) & 0 & 0 \\ 0 & -\frac{1}{3}\left(\sigma_{xx}+\sigma_{yy}+\sigma_{zz}\right) & 0 \\ 0 & 0 & -\frac{1}{3}\left(\sigma_{xx}+\sigma_{yy}+\sigma_{zz}\right) \end{pmatrix} \right\}$$

$$= -\begin{pmatrix} P & 0 & 0 \\ 0 & P & 0 \\ 0 & 0 & P \end{pmatrix} + \left\{ \begin{pmatrix} 2\,\mu\frac{\partial u_x}{\partial x} & \mu(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}) & \mu(\frac{\partial u_x}{\partial z} + \frac{\partial u_z}{\partial x}) \\ \mu(\frac{\partial u_y}{\partial x} + \frac{\partial u_x}{\partial y}) & 2\,\mu\frac{\partial u_y}{\partial y} & \mu(\frac{\partial u_y}{\partial z} + \frac{\partial u_z}{\partial y}) \\ \mu(\frac{\partial u_z}{\partial x} + \frac{\partial u_x}{\partial z}) & \mu(\frac{\partial u_z}{\partial y} + \frac{\partial u_y}{\partial z}) & 2\,\mu\frac{\partial u_z}{\partial z} \end{pmatrix} \right.$$

$$\left. + \begin{pmatrix} -\frac{1}{3}\,(2\,\mu\frac{\partial u_x}{\partial x} + 2\,\mu\frac{\partial u_y}{\partial y} + 2\,\mu\frac{\partial u_z}{\partial z}) & 0 & 0 \\ 0 & -\frac{1}{3}\,(2\,\mu\frac{\partial u_x}{\partial x} + 2\,\mu\frac{\partial u_y}{\partial y} + 2\,\mu\frac{\partial u_z}{\partial z}) & 0 \\ 0 & 0 & -\frac{1}{3}\,(2\,\mu\frac{\partial u_x}{\partial x} + 2\,\mu\frac{\partial u_y}{\partial y} + 2\,\mu\frac{\partial u_z}{\partial z}) \end{pmatrix} \right\}$$

$$= -\begin{pmatrix} P & 0 & 0 \\ 0 & P & 0 \\ 0 & 0 & P \end{pmatrix} + \mu \begin{pmatrix} 2\frac{\partial u_x}{\partial x} & \frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} & \frac{\partial u_x}{\partial z} + \frac{\partial u_z}{\partial x} \\ \frac{\partial u_y}{\partial x} + \frac{\partial u_x}{\partial y} & 2\frac{\partial u_y}{\partial y} & \frac{\partial u_y}{\partial z} + \frac{\partial u_z}{\partial y} \\ \frac{\partial u_z}{\partial x} + \frac{\partial u_x}{\partial z} & \frac{\partial u_z}{\partial y} + \frac{\partial u_y}{\partial z} & 2\frac{\partial u_z}{\partial z} \end{pmatrix} -$$

$$\frac{2}{3}\mu \begin{pmatrix} \frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} + \frac{\partial u_z}{\partial z} & 0 & 0 \\ 0 & \frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} + \frac{\partial u_z}{\partial z} & 0 \\ 0 & 0 & \frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} + \frac{\partial u_z}{\partial z} \end{pmatrix}$$

$$\underbrace{\sigma}_{Stress\ tensor} = \underbrace{-p\mathrm{I}}_{Hydrostatic\ pressure} + \underbrace{\underbrace{\mu(\nabla\vec{\boldsymbol{u}} + (\nabla\vec{\boldsymbol{u}})^T)}_{Viscous\ shear\ stress} - \underbrace{\frac{2}{3}\mu(\nabla.\vec{\boldsymbol{u}})\mathbf{I}}_{Viscous\ normal\ stress}}_{\sigma^D} \qquad \text{Eq. (7)}$$

Knowing that "momentum" is nothing else the product of the mass density and the flow velocity ($\rho\vec{\boldsymbol{u}}$), we solve the integration in Eq. (6) as we did before in the continuity equation. Then, substituting from Eq. (7) gives:

$$\frac{\partial(\rho\vec{\boldsymbol{u}})}{\partial t} + \nabla.(\rho\vec{\boldsymbol{u}} \times \vec{\boldsymbol{u}}) = -\nabla p + \mu(\nabla\vec{\boldsymbol{u}} + (\nabla\vec{\boldsymbol{u}})^T) - \frac{2}{3}\mu\nabla.\left((\nabla.\vec{\boldsymbol{u}})\mathbf{I}\right) + \rho g$$

where **I** is the 3 × 3-dimensional unit matrix.

While,

$$\mu(\nabla\vec{\boldsymbol{u}} + (\nabla\vec{\boldsymbol{u}})^T) = \mu\nabla^2\vec{\boldsymbol{u}} + \mu\nabla(\nabla.\vec{\boldsymbol{u}})$$

where ∇^2 is the Laplacian operator.

Hence,

$$\frac{\partial(\rho\vec{\boldsymbol{u}})}{\partial t} + \underbrace{\nabla.(\rho\vec{\boldsymbol{u}}\otimes\vec{\boldsymbol{u}})}_{Nonlinear\ advection\ term} = \underbrace{-\nabla p}_{Static\ pressure\ term} + \underbrace{\mu\nabla^2\vec{\boldsymbol{u}} + \frac{1}{3}\mu\nabla.\left((\nabla.\vec{\boldsymbol{u}})\right)}_{Viscous\ term} + \underbrace{\rho g}_{\boldsymbol{Volume\ (Body)force}}$$

Eq. (8)

Eq. (8) is among of the most generalized forms for the Navier-Stokes equations written in a vector form.

For an incompressible fluid flow:
ρ is constant, and
the divergence of the velocity is equal to zero. So, we can remove the term $\frac{1}{3}\mu\nabla.\left((\nabla.\vec{\boldsymbol{u}})\right)$ from the viscous force term.

Therefore, Eq. (8) reduces to:

$$\rho\left[\frac{\partial\vec{\boldsymbol{u}}}{\partial t} + \vec{\boldsymbol{u}}.\nabla\vec{\boldsymbol{u}}\right] = -\nabla p + \mu\nabla^2\vec{\boldsymbol{u}} + \rho g$$ Eq. (9)

2.1.3. Application of NSE in a Poiseuille flow between parallel plates

For an incompressible, isothermal and Newtonian fluid flow, Navier-Stokes equations are stated as:

$\nabla.\vec{\boldsymbol{u}} = 0$

$$\rho\left[\frac{\partial\,\vec{\boldsymbol{u}}}{\partial t} + \vec{\boldsymbol{u}}.\nabla\,\vec{\boldsymbol{u}}\right] = -\nabla p + \mu\nabla^2\,\vec{\boldsymbol{u}} + \rho g$$

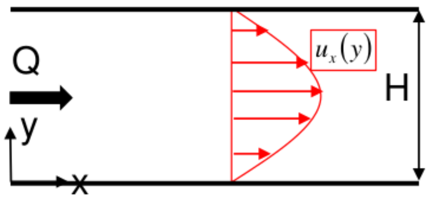


Fig. 3. a Poiseuille flow between parallel plates

Assumptions:

- Incompressible fluid
- Steady-state flow (i.e. $\frac{\partial\,\vec{\boldsymbol{u}}}{\partial t} = 0$)
- Negligible gravity effects (i.e $\rho g \approx 0$)
- Laminar flow (i.e. $u_y = 0$)
- Constant cross-section in X (i.e. $\frac{\partial \boldsymbol{u_x}}{\partial x} = 0$)

These assumptions lead to:

$$-\frac{\partial p}{\partial x} + \mu\,\frac{\partial}{\partial y}\left(\frac{\partial u_x}{\partial y}\right) = 0$$
$$-\frac{\partial p}{\partial y} = 0$$

Boundary conditions are defined as:

$u_x = 0$ at y =0
$u_x = 0$ at y =H

Hence,

$$u_x(\mathrm{y}) = \frac{1}{2\mu}\,\mathrm{y}\,(\mathrm{y} - \mathrm{H})\,\frac{\Delta p}{L}$$

Each of rate, maximum velocity and average velocity can be calculated as follows:

$$Q = -\frac{H^3}{12\mu}\frac{\Delta p}{L}$$

$$u_{max} = -\frac{H^2}{8\mu}\frac{\Delta p}{L}$$

$$\overline{u} = -\frac{H^2}{12\mu}\frac{\Delta p}{L}$$

2.1.4. Application of NSE in a Poiseuille flow in a cylinder

For an incompressible, isothermal and Newtonian fluid flow, Navier-Stokes equations are stated as:

$$\nabla . \vec{\boldsymbol{u}} = 0$$
$$\rho \left[\frac{\partial\, \vec{\boldsymbol{u}}}{\partial t} + \vec{\boldsymbol{u}}.\nabla\, \vec{\boldsymbol{u}}\right] = -\nabla p + \mu \nabla^2\, \vec{\boldsymbol{u}} + \rho g$$

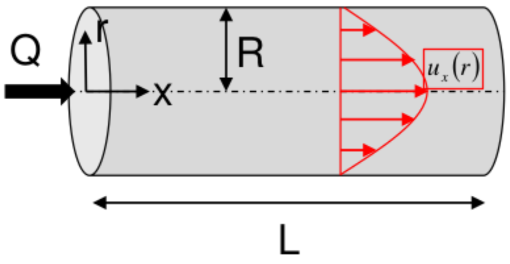


Fig. 4. a Poiseuille flow in a cylinder

Assumptions:

- Incompressible fluid
- Steady-state flow (i.e. $\frac{\partial\, \vec{\boldsymbol{u}}}{\partial t} = 0$)
- Negligible gravity effects (i.e. $\rho g \approx 0$)
- 1-D radial symmetry
- Laminar flow (i.e. $u_r = 0$)
- Constant cross-section in X (i.e. $\frac{\partial \boldsymbol{u_x}}{\partial x} = 0$)

These assumptions lead to:

$$-\frac{\partial p}{\partial x} + \mu \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial u_x}{\partial r}\right) = 0$$
$$-\frac{\partial p}{\partial r} = 0$$

Boundary conditions are defined as:

$u_x = 0$ at r =R

Hence,

$$u_x(\mathrm{r}) = \frac{1}{4\mu}\,(r^2 - R^2)\,\frac{\Delta p}{L}$$

Each of rate, maximum velocity and average velocity can be calculated as follows:

$$Q = -\frac{\pi R^4}{8\mu}\frac{\Delta p}{L}$$

$$u_{max} = -\frac{R^2}{4\mu}\frac{\Delta p}{L}$$

$$\overline{u} = -\frac{R^2}{8\mu}\frac{\Delta p}{L}$$

2.1.5. Dimensionless incompressible NSE

As a standard procedure, when we deal with differential equations, the first step is often to redefine them into a dimensionless form. And there are many reasons behind that. “Small” and “large” are meaningless for dimensional values; because it is always possible to change the system of units used. Also, nature knows no units.
Dimensionless Navier-Stokes equations are considered a powerful tool that can be used to generalize the fluid flow behavior into nearly all systems and domains. They are even especially suited for mathematical analysis and numerical simulation which are mainly based on dimensionless equations.

To derive the dimensionless Navier-Stokes equations, the following quantities/scaling parameters

- U [m/s]– the characteristic flow velocity scale
- L [m]– the characteristic flow length scale

are introduced.

By rescaling the space and time variables, and even the gradient operator, one obtains:

$$\vec{\boldsymbol{u}}^* = \frac{\vec{\boldsymbol{u}}}{U}, \quad p^* = p\,\frac{1}{\rho U^2}, \quad g^* = g\,\frac{L}{U^2}, \quad \frac{\partial}{\partial t^*} = \frac{L}{U}\frac{\partial}{\partial t}, \quad \nabla^* = L\nabla$$

Rearranging the above relations and inserting into Eq. (9) to obtain:

$$\rho\left[\frac{U^2}{L}\frac{\partial \vec{\boldsymbol{u}}^*}{\partial t^*} + U\,\vec{\boldsymbol{u}}^*.\frac{\nabla^*}{L}\,(U\,\vec{\boldsymbol{u}}^*)\right] = -\,\frac{\nabla^*}{L}\,p^*\,\rho U^2 + \mu\,\frac{\nabla^{*2}}{L^2}\,U\,\vec{\boldsymbol{u}}^* + \rho\,\frac{g_*\,U^2}{L}$$

Dividing both sides by $\rho\frac{U^2}{L}$ yields:

$$\frac{\partial \vec{\boldsymbol{u}}^*}{\partial t^*} + \underbrace{\vec{\boldsymbol{u}}^*.\nabla^*\vec{\boldsymbol{u}}^*}_{Advection\ term} = \underbrace{-\nabla^*\,p^*}_{\substack{Static\\ pressure\\ term}} + \underbrace{\overbrace{\frac{\mu}{\rho U L}}^{\frac{1}{R_e}}\,\nabla^{*2}\,\vec{\boldsymbol{u}}^* +}_{\substack{Viscous\\ term}} \underbrace{g^*}_{Body\ force\ term} \qquad \text{Eq. (10)}$$

Now, it is clear that R_e is the only parameter of the problem and describes the contribution of viscosity:

- Low R_e flows are dominated by viscosity and remain laminar
- In high R_e flows, viscosity effects are limited to very narrow regions along walls (boundary layers), or along boundaries of different flow streams (shear layers). Such flows are always highly turbulent.

Therefore, we can easily identify the relative importance of each part of the equation. When there is no exact solution -and this is often the case-, we can neglect the terms which are significantly small in comparison with the others, therefore leading to appropriate solutions for complex problems.

2.2. Finite volume method

The finite volume method (FVM) is a discretization technique used for the numerical simulation of partial differential equations of various types (for instance, elliptic, hyperbolic or parabolic) (Eymard et al., 2019). Generally, it is well-suited for those PDE arising from physical conservation laws (typically, the Navier-Stokes equations and the turbulence equations) (Chen, 2009). It had been firstly introduced into the field of computational fluid dynamics (CFD) in the beginning of the 1970s "McDonald 1971, MacCormack and Paullay 1972" (Kolditz, 2002). FVM solvers are among the most popular ones in CFD, being implemented in numerous commercial software packages like *Phoenics, ANSYS Fluent*, *SOLIDWORKSFloWorks and Flow-3D,* and non-commercial packages like *OpenFOAM* (Rapp, 2017).

In the finite volume method, the domain of interest is first discretized into a number of equally spaced nodes surrounded by non-overlapping finite volumes (control volumes) (Neill et al., 2018). Next, unlike the finite element method and the finite difference method, the conservation laws are applied to each individual control volume in an integral form (Reynolds transport theorem) rather than in a partial differential form. That should lead in the end to enough linear algebraic equations which can be solved numerically to compute the unknown physical parameters at each node.

In general, we distinguish between the two approaches adopted for the discretization of the computational domain into finite volumes (control volumes): cell-centered approach and cell-vertex approach (Kolditz, 2002). In the cell-centered approach, the control volumes coincide with the mesh cells. While in the cell-vertex approach, each node of the mesh is the center of a control volume, whose boundaries are assigned by connecting the centroids of each cell and the midpoints of each cell edge as depicted in figures (5), (6) for a two-dimensional case.

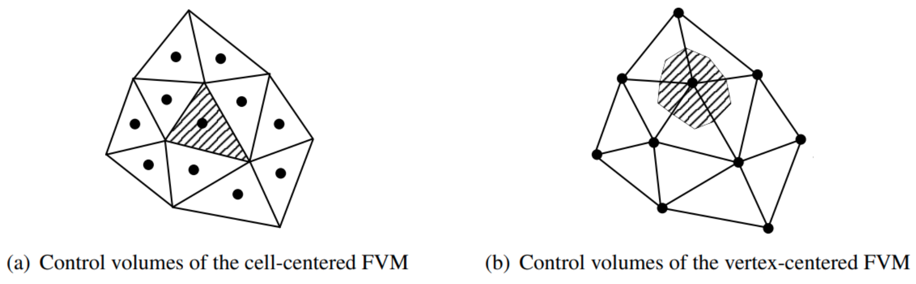

(a) Control volumes of the cell-centered FVM
(b) Control volumes of the vertex-centered FVM

Fig. 5. Meshes and control volumes of two types of FVM. The unknowns are associated to the black nodes (Chen, 2009).

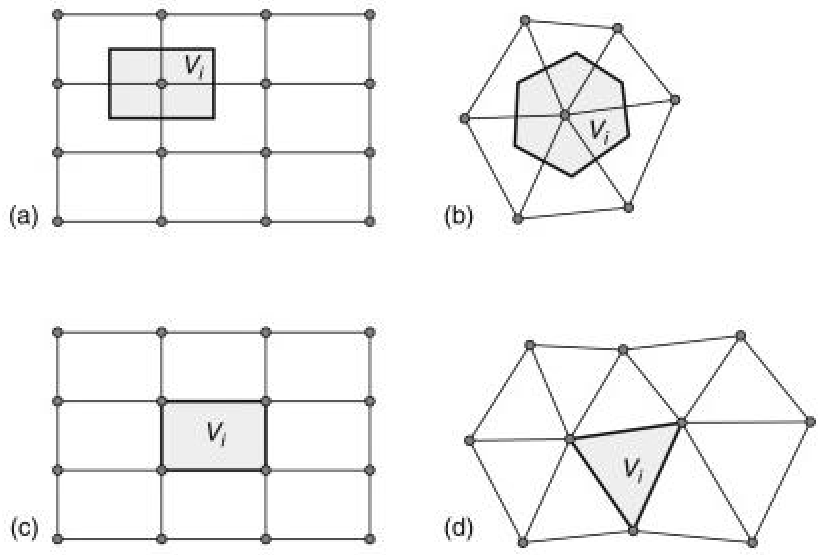


Fig. 6. Mesh and dual mesh in vertex-centred FVM (a,b) and cell-centred FVM (c,d). Control volumes are defined by the grey-coloured areas (Aleksendrić et al., 2015)

The finite volume method is a bit different from (but sometimes related to) the finite element method which or the finite difference method (Eymard et al., 2019). In contrast to FEM which is based on the discretization of the weak formulation and FDM which is based on the classic formulation, FVM is based on the discretization of the balance equation.

Since the FVM is based on the discretization of the balance equation directly in space, an obvious virtue of it is the conservation of quantities, such as mass, momentum and energy, on a discretized (local) level, i.e. the flux entering a given cell is identical to that leaving the adjacent one over the entire computational domain. In other words, that is to say one cell's loss is always another cell's gain. This feature makes it is extensively used in several fundamental engineering fields in general and quite attractive when modelling problems where the flux is of great importance, such as fluid mechanics, mass transfer, heat transfer, semi-conductor device simulation and oil recovery simulations. Experience has proved that conservative schemes are generally more accurate and stable than non-conservative ones, particularly for strong gradients (large derivatives) inside a certain domain (Kolditz, 2002).

As flexible as the FEM, the finite volume method can handle any type of mesh; structured and unstructured, (Aleksendrić et al., 2015) leading to robust schemes. Structured meshes are often comprised of orthogonal quadrilaterals (2D) or hexahedrons (3D) that follow a uniform pattern. Unstructured meshes do not follow a uniform pattern, but are arbitrary combinations of triangles, quadrilaterals (2D), and tetrahedrons, hexahedrons or pyramids (3D). These unstructured meshes take full advantage of modeling complex and irregular geometry configurations. The FVM, therefore, is more flexible and adequate than the finite difference method where the latter is mainly defined based on structured grids, simple domains and homogeneous geometries. That is why, from the numerical point of view, the FVM is a generalization of the FDM in terms of geometry and topology because simple and uniform finite volume schemes can be reduced to finite difference ones (Kolditz, 2002).

Roughly speaking, given a number of discretization points defined by a mesh, the finite difference method is based on assigning one discrete unknown and writing one equation per each discretization point. Then, the derivatives of that unknown are replaced by finite differences using Taylor expansion (Eymard et al., 2019). However, it becomes difficult to use the finite difference method in case the coefficients involved in the equation suffer discontinuity (e.g., in case of a heterogeneous medium). With the finite volume method, such discontinuity of the coefficients will not be a problem provided the mesh is chosen in a way that these discontinuities occur on the boundaries of the control volumes (Eymard et al., 2019).

From an industrial perspective, the finite volume method has been found a cheap and robust technique for the discretization of conservation laws (Eymard et al., 2019). It is cheap thanks to its advantage in terms of memory storage requirement and solution speed providing short and reliable computational coding for complex problems, such as high Reynold number turbulent flows and source-term dominated flows (e.g. combustion) (Patankar, 1980). However, in some cases, it becomes difficult to design schemes with a certain given precision. Indeed, the finite element method can be much more precise than the finite volume method when using higher order polynomials, but it requires an appropriate functional framework which is not always available in industrial problems. Other more precise methods include, for instance, spectral methods or particle methods but these methods can be more expensive and less robust than the finite volume method. By robust, we mean a scheme that works well even for relatively difficult equations, e.g. nonlinear systems of hyperbolic equations, and can easily be extended to more physical and realistic contexts than the classical academic problems) (Eymard et al., 2019).

2.3. Lagrangian and Eulerian

2.3.1. Overview

Describing the fluid motion has always been an active research topic in the field of CFD numerical simulations. To do so, one needs to know the variations of physical quantities such as, density, velocity, pressure, temperature, stresses, etc., as functions of time, everywhere within a certain spatial region. Generally, there are two ways to describe the fluid motion in CFD simulations, the Lagrangian and the Eulerian method. In fluid dynamics, both have been used with considerable success, each with its own advantages and disadvantages (Mei, 2001).

2.3.2. Lagrangian Method (surface tracking method)

It considers particles as a discrete phase, tracks all of them and describes the variations around each individual one along its own trajectory (Mei, 2001). In Lagrangian methods, the grid can be configured to fit the shape of the interface, and thus adapted continually to it. The two main advantages of Lagrangian methods are: (i) explicitly tracking the interface, (ii) applying interfacial boundary conditions at the exact position of the interface. However, the main disadvantage with Lagrangian methods is the lack of the numerical accuracy due to that irregular grid (Francois, 1998).

2.3.3. Eulerian Method (volume tracking method)

In Eulerian methods, variations in physical quantities are described at fixed points as a function of time, i.e. different particles pass the same point at different times (Mei, 2001). The fluid particle is treated as a continuum where its conservation equations are developed on a control volume basis and in a similar form as that for the fluid phase itself. In contrast to Lagrangian methods, a fixed grid is employed, and the fluid interface is not explicitly tracked but reconstructed from appropriate field variables, such as the volume fluid fraction. That is why further procedures are needed to deduce the interface position from the volume fluid fraction. Also, these interfacial boundary conditions will have to be manipulated to be shown in the governing transport equation (Francois, 1998). The main advantage of Eulerian methods is the ability to accurately calculate the field variables due to their fixed grid. However, their two major disadvantages are: (i) the position of the interface is not handled with a high precision (ii) the possible smearing of the discontinuous surfaces; due to the manipulation of the interfacial boundary conditions (Francois, 1998).

For multiphase flows, it is necessary in Eulerian methods to compute the flow of fluid through the whole mesh which results in averaging the flow properties, such as density, viscosity, ... etc. This "averaging process" leads to smoothing of all variations in flow quantities, especially smearing surfaces of discontinuity such as free surfaces. Thus, some special treatment is required to overcome this loss in resolution within the interface, recognize the discontinuity and avoid averaging (Hirt et al., 1979).

If we know the position of each particle as a function of time, we can transform the conservation laws from a Lagrangian system into an Eulerian system through Taylor expansion, as follows:

$$f(t+\Delta t, x_0+\vec{\boldsymbol{u}}\Delta t) \approx f(t, x_0) + \frac{\partial f}{\partial t}\Delta t + \sum_{i=1}^{3}\frac{\partial f}{\partial x_i}\,u_i\,\Delta t$$

$$\frac{f(t+\Delta t, x_0+\vec{\boldsymbol{u}}\Delta t) - f(t, x_0)}{\Delta t} \approx \frac{\partial f}{\partial t} + \sum_{i=1}^{3}\frac{\partial f}{\partial x_i}u_i$$

$$\frac{Df}{Dt} = \frac{\partial f}{\partial t} + \nabla f.\vec{\boldsymbol{u}} = \underbrace{\left(\frac{\partial}{\partial t} + \vec{\boldsymbol{u}}.\nabla\right)f}_{\frac{\boldsymbol{D}()}{\boldsymbol{Dt}}}$$

$$\underbrace{\frac{D(\,)}{Dt}}_{\substack{Lagrangian\ derivative\\ (or\ material\ derivative)}} = \overbrace{\underbrace{\frac{\partial(\,)}{\partial t}}_{\substack{Local\ derivative\\ (Eulerian\ derivative)}}}^{Unsteady\ effects} + \overbrace{\underbrace{\vec{\boldsymbol{u}}}_{\substack{How\ quickly\\ fluid\ changes\\ position}} \quad . \quad \underbrace{\nabla(\,)}_{\substack{Rate\ at\ which\\ propperty\ changes\\ with\ position}}}^{Convective\ effects}$$

2.3.4. NSE in Lagrangian formulation

2.3.4.1. Continuity equation

$$\underbrace{\frac{\partial\rho}{\partial t} + \nabla.(\rho\vec{\boldsymbol{u}}) = 0}_{Eulerian} \xRightarrow{\nabla.(\rho\vec{\boldsymbol{u}}) = \rho\nabla.\vec{\boldsymbol{u}} + \vec{\boldsymbol{u}}.\nabla\rho} \underbrace{\frac{D\rho}{Dt} + \rho\nabla.\vec{\boldsymbol{u}} = 0}_{Lagrangian}$$

2.3.4.2. Momentum equation

$$\underbrace{\frac{\partial(\rho\vec{\boldsymbol{u}})}{\partial t}+\vec{\boldsymbol{u}}.\boldsymbol{\nabla}(\boldsymbol{\rho}\vec{\boldsymbol{u}})=-\nabla \mathrm{p}+\mu\nabla^{2}\vec{\boldsymbol{u}}+\frac{1}{3}\mu\nabla.\left((\nabla.\vec{\boldsymbol{u}})\right)+\rho g}_{Eulerian}\xRightarrow{\frac{D(\ )}{Dt}=\frac{\partial(\ )}{\partial t}+\vec{\boldsymbol{u}}.\nabla(\ )}$$

$$\underbrace{\frac{D\,((\rho\vec{\boldsymbol{u}})}{Dt}=-\nabla \mathrm{p}+\mu\nabla^{2}\vec{\boldsymbol{u}}+\frac{1}{3}\mu\nabla.\left((\nabla.\vec{\boldsymbol{u}})\right)+\rho g}_{Lagrangian}$$

Navier-Stokes equations in Largangian formulation become:

$$\frac{D\rho}{Dt}+\rho\nabla.\vec{\boldsymbol{u}}=0 \qquad \text{Eq. (11)}$$

$$\frac{D\,(\rho\vec{\boldsymbol{u}})}{Dt}=-\nabla \mathrm{p}+\mu\nabla^{2}\vec{\boldsymbol{u}}+\frac{1}{3}\mu\nabla.\left((\nabla.\vec{\boldsymbol{u}})\right)+\rho g \qquad \text{Eq. (12)}$$

Finally, choosing which type of methods to use highly depends on the objective and characteristics of the problem under examination (Zhang et al., 2007). For example, if the interface details are unlikely to have an impact on the flow features, Eulerian methods are more attractive. On the other hand, if the discontinuity nature across the interface is to be simulated with a high precision, Lagrangian methods could be preferable (Francois, 1998). Moreover, there are also combined Eulerian-Lagrangian methods yielding solutions with combined characteristics of both Eulerian and Lagrangian methods.

2.4. Volume-of-fluid (VOF) method

A free surface is defined as a surface on which discontinuities arise in one or more variables (Hirt et al., 1979). In CFD, therefore, the free surface is the interface separating between two immiscible fluids. A simple yet powerful method which is applicable to both two- and three-dimensional computations, is the volume of fluid method (VOF), first introduced by Nichols and Hirt in 1981. Due to its robustness and ease of implementation and parallelization together with its ability to conserve mass and to produce reasonably sharp interfaces (Hoang et al., 2013), VOF is very popular in multiphase simulations and has been implemented in both commercial (ANSYS Fluent and Flow-3D) and open-source (OpenFOAM) CFD packages. It is based on the concept of the fractional volume of fluid (α) in each cell of the computational domain, where α is a continuous function with $0 \leq \alpha \leq 1$. A volume fraction value of 1 indicates that the cell is completely filled with phase A and a volume fraction of 0, indicates that the cell is completely filled with phase B. The values between 0 and 1 represent the interface between the two fluids.

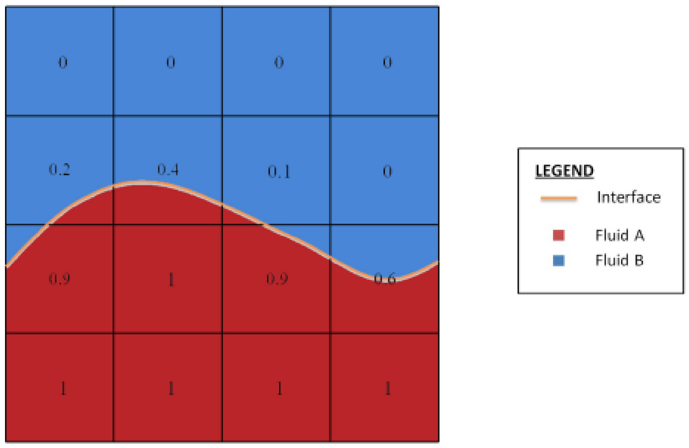


Fig. 7. Volume of fluid approach (Haider, 2013)

Classified as an Eulerian method, the VOF method has naturally shown its strength to deal with highly non-linear problems characterized by large interface movement and topological changes (Hoang et al., 2013). It has been proved to be more flexible and efficient than the other methods for dealing with complicated interfaces (Hirt et al., 1979). It is also considered computationally friendly with minimum storage requirements, because it introduces only one additional storage variable in each mesh cell to define the interface.

This makes it consistent with the storage requirements for all other dependent variables, such as pressure, temperature, density ... etc., since it is customary to use only one numerical value for each dependent property defining the fluid state in each grid cell. That is why its conservative use of storage is highly advantageous from the economic point of view, especially when applied to three-dimensional computations (Hirt et al., 1979).

By definition, it is necessary for the VOF method in multiphase flows to compute the flow of fluid through the whole mesh, and this requires averaging the flow properties. Thus, the density and viscosity are averaged as follows, respectively (Hirt et al. 1979):

$$\rho = \alpha\, \rho_1 + (1 - \alpha)\, \rho_2 \qquad \text{Eq. (13)}$$

$$\mu = \alpha\, \mu_1 + (1 - \alpha)\, \mu_2 \qquad \text{Eq. (14)}$$

1,2 denote the two immiscible phases

This "averaging process", however, is one of the biggest drawbacks of the VOF method, like all other Eulerian methods. That is because it leads to smoothing all variations in the fluid flow quantities, and, particularly, a smearing of surfaces of discontinuity such as free surfaces. Consequently, interfaces lose their definition, and the volume fractional variable loses its supposed discontinuous nature. One method proposed to avoid averaging across the free surface and thus overcome this loss in resolution and recover its shape, is advecting the volume fluid fraction with the local fluid flow velocity (Hirt et al. 1979).

The governing equation for α, referred as the volume fraction equation or the transport equation is:

$$\frac{\partial \alpha}{\partial t} + \nabla.(\alpha\, \vec{\boldsymbol{u}}) = 0 \qquad \text{Eq. (15)}$$

Another disadvantage of the VOF method is the presence of non-physical spurious currents. The reason behind them is that the curvature interface is not handled with high precision, originating from the fact that the interface is not explicitly tracked or defined but reconstructed based on the fractional volume of the fluid (Hoang et al. 2013). Only in 1992, a technique, referred to as the Continuum Surface Force (CSF) model, has been developed to impose the surface tension effects in a more efficient manner. The smaller the error in surface tension calculations, the less significant spurious currents are induced.

2.5. Continuum Surface Force (CSF)

Although they possess an elastic skin, liquid surfaces are in a state of tension due to the uneven molecular forces of attraction exerting on the fluid molecules at or near the surface (Brackbill et al., 1992). Both surface tension and contact angle arise from these interaction forces between the particles of these two different immiscible fluids (Raeini et al., 2012). These abrupt changes in molecular forces taking place through an interface when fluid properties change discontinuously, can easily show how much the surface tension is an inherent characteristic of material interfaces. Surface tension results in a localized, microscopic "surface force" exerting itself on fluid elements at interfaces in both directions: normal and tangential. Fluid interfacial motion induced by surface tension plays a fundamental role in many natural and industrial phenomena such as: capillarity studies, hydrodynamic stability, low-gravity fluid flow, cavitation, surfactant behavior and droplet dynamics in clouds and in fuel sprays utilized in internal combustion engines. Typically, numerical models are involved in the analysis of such processes as a means to understand the resulting nonlinear fluid flows (Brackbill et al., 1992).

Modelling multiphase flows in porous media concerns forces acting at different scales: viscous, capillary and wall adhesion. At the pore scale, which is of our interest here, the capillary forces are more significant than the viscous ones in most transport problems. For example, capillary numbers in oil and gas reservoirs are typically in the range $N_c = \mu.u_d/\sigma = 10^{-10}$ to 10^{-5} ; where μ is viscosity, u_d is the Darcy velocity and σ is the interfacial tension. Consequently, any small errors in the computation of the surface tension, if not handled carefully, will lead to errors in the numerical calculations of the capillary forces and will introduce instabilities in the numerical simulation up to a point where it has no predictive capability (Raeini et al., 2012). Moreover, these errors in the surface tension computation will cause Front-Capturing methods (such as Volume-of-Fluid and Level-Set method) to induce non-physical spurious currents at the interface (Inguva et al., 2020).

However, previous methods have suffered from difficulties in modeling interfaces with complex topologies. A method, Continuum Surface Force, has been developed where interfaces between fluids of different properties are represented as transition regions of finite thickness. A force density is defined at each point in that transition region, which is proportional to the curvature of the surface at each point.

$\mathbf{f_s} = 2\ \sigma\ \kappa\ \nabla\alpha$ Eq. (16)

where
$\mathbf{f_s}$ is the force term generated due to the interfacial tension
σ is the interfacial tension
κ is the interfacial curvature:

$\kappa = \frac{1}{2}\nabla\ .\ \hat{\boldsymbol{n}} = \frac{1}{2}\nabla\ .\ \frac{n}{|n|} = \frac{1}{2}\nabla\ .\ (\frac{\nabla\alpha}{|\nabla\alpha|})$ Eq. (17)

It is normalized to recover the conventional description of surface tension on an interface when the ratio of local transition region thickness to local radius of curvature approaches zero (Brackbill et al., 1992).

The Continuum Surface Force (CSF) is the most commonly used surface tension force model. It is implemented in many CFD packages, such as: OpenFOAM, Fluent and StarCCM+ (Vachaparambil et al. 2019). It alleviates these previous interface topology constraints without sacrificing accuracy, simplifies the calculation of surface tension, enables accurate modeling of two- and three-dimensional fluid flows driven by surface forces, eliminates the need for interface reconstruction, and does not impose any modeling restrictions on the complexity, number or dynamic evolution of the fluid interfaces. One more of its characteristics is being perfectly suited for Eulerian interfaces which are not generally aligned with the computational grid. In addition, the CSF model has been found successfully applicable to a large number of fluid phenomena influenced by the interfacial surface tension with similar mathematical structure and many new and physically interesting problems, such as modelling incompressible fluid flow in low-gravity environments, droplet dynamics and capillarity. Even, it has been validated on both static and dynamic interfaces (Brackbill et al., 1992).

In fact, the explicit time step constraint is often more restrictive than other constraints. The continuum formulation, however, does not increase the severity of this constraint but may enable us to formulate implicit equations. This implicit formulation which removes that constraint would decrease the cost of surface tension calculations by enormous factors. Unfortunately, the nonlinearity and complexity of surface tension equations still make it a challenging problem. So clearly, there can be improvements in the numerical implementation of the method even though it has been verified that reformulating a discontinuous interface problem as a continuum problem has demonstrably improved describing surface tension phenomena (Brackbill et al., 1992).

2.6. OpenFOAM

OpenFOAM is a three-dimensional CFD tool and a multiphasic simulation platform mainly devoted to fluid flow and based on a finite-volume code (Soulaine, 2018). It was primarily developed by Imperial Collage London in 1989. In 1996, its first version was lunched. Since 2004, it has been released under GPL license owned by OpenCFD Ltd. It is a freely available open-source package whose toolbox is written by C++ (object-oriented programming) and can be easily downloaded at www.openfoam.org. "OpenFOAM" stands for "**Open** **F**ield **O**peration **A**nd **M**anipulation". It is used over a wide variety of engineering and science areas by both commercial and academic organisations. In contrast to the other commercial tools like Fluent and StarCCM+ which require users to abide by the frameworks of the software, OpenFOAM has the greatest flexibility due to the availability to access and modify its source code (Inguva et al., 2020). It provides friendly syntax for partial differential equations solving. Its parallel computation is implemented at the lowest level. It is characterized by a cross-platform installation, i.e. working on both Windows and Linux. It has a huge solver database covering the breadth and depth of CFD.

OpenFOAM consists of more than 200 programs and is not only one executable. Its utilities are subdivided into three categories:

1- Pre-processing
   - Meshing (blockMesh, snappyHexMesh, foamHexMesh, …)
   - Mesh conversion generated by a third-part meshing tool (Ansys, ideas, CFX, Gambit, Salomé, Gmsh, Gambit, …)

2- Solvers
   a. incompressible flow solvers
   b. compressible flow solvers
   c. multiphase flow solvers (VOF, Euler-Euler, …)
   d. Particle-tracking solvers
   e. Buoyancy-driven flow solvers
   f. Solid mechanics solvers
   g. Solvers for combustion problems
   h. Solvers for heat transfer problems
   i. Molecular dynamic solvers
   j. Electro-magnetic solvers
   k. Turbulence approach solvers like DNS, LES and RANS…
   l. Direct Simulation Monte Carlo solvers

In addition to these standard built-in solvers, OpenFOAM's syntax easily lends itself to create custom solvers and boundary fields.

3- Post-processing

a. Distribution to visualize the modelling process with *ParaView* (or the more famous *paraFoam*)

b. Exporting to other post-treatment softwares; such as Fluent, EnSight, Fieldview, Tecplot, Mayavi …

c. *postprocess* command for 1-D or 2-D sampling (exporting to gnuplot, Grace/xmgr et jPlot)

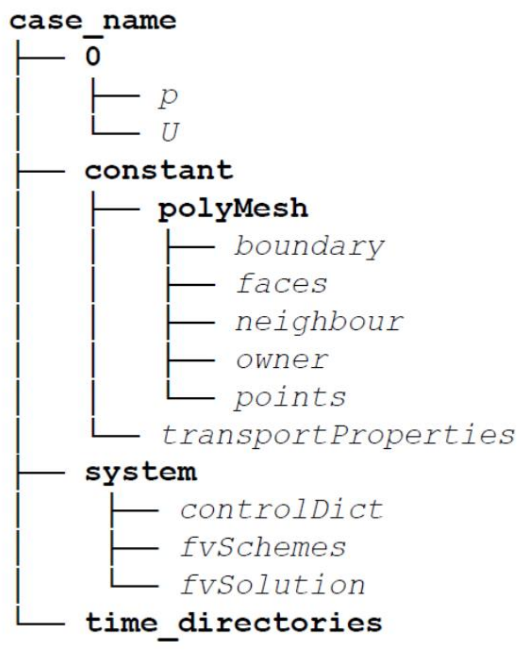
PROS

- Completely-free (unlimited license)
- Regularly updated every 6 months
- An extra tool for code-to-code benchmarks
- Availability of several out-of-the-box solvers and their tutorials
- Simplicity to program partial differential equations
- An important and highly reactive community (summer schools, online forum, conference)
- Providing a direct access to the source code (it is not a black-box)

CONS

- Need time and effort to learn.
- Lack of literature documentation
- Too many forks
- No presence for an official GUI
- Need to deal with the Unix command lines and C++ programming

General structure of an OpenFOAM case



```
case_name
├── 0
│   ├── p
│   └── U
├── constant
│   ├── polyMesh
│   │   ├── boundary
│   │   ├── faces
│   │   ├── neighbour
│   │   ├── owner
│   │   └── points
│   └── transportProperties
├── system
│    ├── controlDict
│    ├── fvSchemes
│    └── fvSolution
└── time_directories
```

Fig. 8. General structure of an OpenFOAM case

Firstly, according to the solver or application we would like to use, we will need different files in each subdirectory.

`0:`

- contains initial conditions (IC) and boundary conditions (BC).

`constant:`

- contains all the physical properties with constant values (e.g., turbulence modeling properties, advanced physics, thermodynamic properties, transport properties … etc.
- The sub-directory *polyMesh* includes all the information regarding the grid.

`system:`

- contains the simulation setup settings, run-time control and solver numerics (e.g., choice of the linear solver, of the time step, of the discretization schemes, the output files … etc).

<u>`time directories:`</u>

- one folder per each time step. In each folder, there are as many files as the computed fields (U, p, T, k, Omega, $Y_{i, \dots}$)

Chapter 3: Methodology

In this work, using OpenFOAM v8, we simulate the relaxation of a 2D, stationary, square oil droplet immersed in a continuous water phase to its final circular shape, in the absence of the gravity effect. The densities of water and oil are 1000 kg/m^3 and 500 kg/m^3 respectively, the viscosities of water and oil are 0.001 Pa.s and 0.0025 Pa.s respectively and the surface tension between both fluids is 0.0236 N/m. The diameter of the relaxed droplet is set to 2R = 338.52 μm, which reasonably represents the typical dimension of a segmented flow in a microfluidic system. The computational domain size is (600 × 600) μm^2, corresponding to 100 ×100 cells. (Therefore, the grid cell size is Δx=Δy = 6 μm)

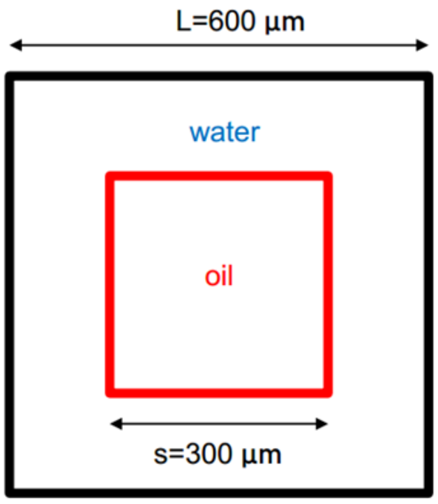


Fig. 9. A stationary droplet case study (before relaxation)

$$A_{square} = A_{circle}$$

$$S^2 = \pi R^2$$

$$R = \frac{S}{\sqrt{\pi}} = \frac{300}{\sqrt{\pi}} = 169.26\ \mu\text{m}$$

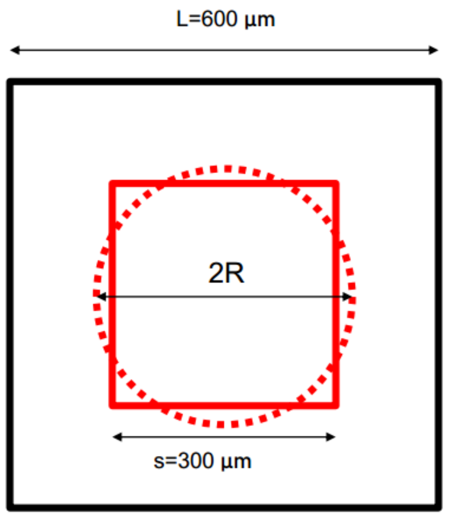


Fig. 10. Relaxation of the static droplet

In OpenFOAM, a single set of Navier–Stokes equations is discretized on the basis of a finite-volume approach and solved simultaneously with the transport equation for the VOF index function, α. Considering Newtonian, incompressible, and immiscible two fluids (water and oil), the governing equations can be formulated as:

$$\nabla . \vec{\boldsymbol{u}} = \mathbf{0}$$

$$\rho \left[\frac{\partial \vec{\boldsymbol{u}}}{\partial t} + \vec{\boldsymbol{u}} . \nabla \vec{\boldsymbol{u}}\right] = -\nabla p + \mu \nabla^2 \vec{\boldsymbol{u}} + \rho g + \mathbf{f_s}$$

$$\frac{\partial \alpha}{\partial t} + \nabla . (\alpha u) = 0$$

Weighted by the VOF index function (α), the density and viscosity are averaged over the two phases as follows, respectively:

$$\rho = \alpha \, \rho_1 + (1 - \alpha) \, \rho_2$$

$$\mu = \alpha \, \mu_1 + (1 - \alpha) \, \mu_2$$

Subscripts 1 and 2 denote water and oil phases, respectively.

$\mathbf{f_s}$ is the force term generated due to the interfacial tension, modeled as a volumetric force applying the Continuum Surface Force (CSF) method:

$$\mathbf{f_s} = 2 \, \sigma \, \kappa \, \nabla \alpha$$

where σ is the interfacial tension, and κ is the interfacial curvature:

$$\kappa = \frac{1}{2} \nabla . \hat{\boldsymbol{n}} = \frac{1}{2} \nabla . \frac{n}{|n|} = \frac{1}{2} \nabla . (\frac{\nabla \alpha}{|\nabla \alpha|})$$

In order to obtain a sharp fluid interface, an artificial compression velocity term $\nabla . [u_r \alpha (1 - \alpha)]$ is introduced in the VOF index function, to become:

$$\frac{\partial \alpha}{\partial t} + \nabla . (\alpha \vec{\boldsymbol{u}}) + \nabla . [u_r \alpha (1 - \alpha)] = 0$$

where u_r is the relative compression velocity, defined as the velocity of the phase 1 (the wetting fluid) minus the velocity of the phase 0 (the non-wetting fluid); $u_r = u_w - u_{nw}$, and evaluated as follows:

$$u_r = n_f \min \left[C_{\propto} \frac{|F|}{|S_f|}, \; \max\left(\frac{|F|}{|S_f|}\right) \right]$$

where

n_f is the normal vector to the cell face

$C_\propto$ is an adjustable coefficient to control the level of compression, called "`cAlpha`" in OpenFOAM within the range 0 < `cAlpha` < 2

F is the volume flux across the cell face

S_f is the cell surface area

The term $\alpha(1-\alpha)$ guarantees that this equation is only active in the interface area where $0 < \alpha < 1$

Thus, we can evaluate the influence of the interface sharpening coefficient ($C_\propto$) when `cAlpha = 0, 1 and 2`

To reduce the magnitude of spurious currents arising from the numerical errors in the surface tension calculations due to using the VOF method, the VOF index function is smoothed by twice applying the 'Laplacian filter' proposed by Lafaurie et al., 1994. Since this smoother is not implemented in the OpenFOAM library, an external code found in literature is used and linked to the VOF solver. The smoothed index function $\tilde{\alpha}$ is given by:

$$\tilde{\alpha}_p = \frac{\sum_{f=1}^{n} \alpha_f S_f}{\sum_{f=1}^{n} S_f}$$

where the subscript P refers to the cell index and f refers to the face index. The interpolated value of α_f at the face centre is computed using linear interpolation.

Chapter 4: Simulation steps and results

1- Using the `blockMesh` utility, a default multi-hexahedral block mesh generator of OpenFOAM, and according to the dictionary file named `blockMeshDict` located in the `system` directory, our computational domain is discretized over a two-dimensional mesh. Because the number of mesh cells in z-direction is one, and also in the boundary section, the surface along the z-direction (`frontAndBack`) is defined as empty. The generated hex block is defined by eight vertices, in a sequential order, considering a given mesh density (i.e. number of cells) and a uniform mesh stretching in each direction. Edges are set to be straight by default.

2- Assigning the oil droplet in its initial square shape to the domain by defining the box region in the `setFieldsDict` dictionary, also located in the `system` directory, and then using the `setFields` utility.

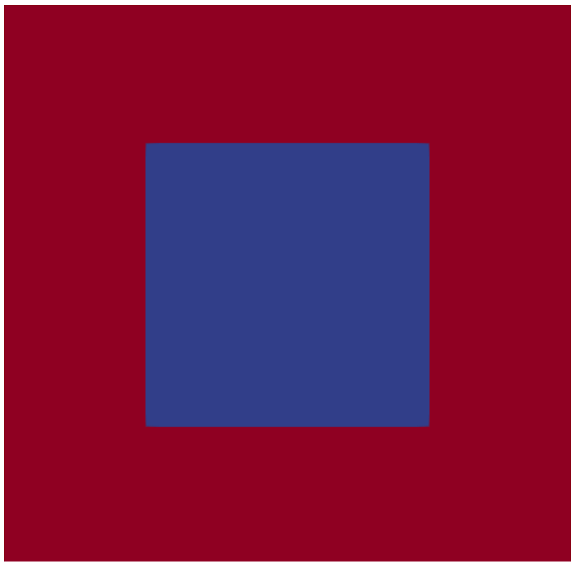


Fig. 11. Simulation domain used. Assigning the oil droplet (blue) into the continuous water phase (Red)

3- In order to minimize the inter-processors communication and the processor workload **and subsequently** reasonably decrease the computational time and the numerical cost, we decompose our domain into 4 sub-domains (refer to the number of available processors among which our case will be distributed) using the `decomposePar` utility which reads the file dictionary `decomposeParDict`, also located in the `system` directory. We used `scotch` as a decomposition method in our case, which requires no geometric input from the user and attempts to minimize the number of processor boundaries.

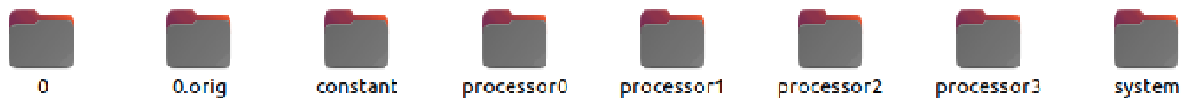


Fig. 12. Decomposition the computational domain over 4 processors

4- After decomposition, we run the standard VOF solver available within OpenFoam, *interFoam*, in parallel using the public domain *openMPI* implementation of the standard message passing interface (MPI).

5- For post-processing purposes, we reconstruct the mesh and field data to recreate the complete domain and fields by executing `reconstructPar` without any additional options where the sets of time directories stored in each processor directory are merged into a single set of time directories being saved in the main case directory.

6- We run `paraFoam` to visualize our simulation results

7- We extract the centroids coordinates at the latest time ($t = 0.1$), for the three cases typing the following command:

```
postProcess -func writeCellCentres -latestTime
```

8- We plot the distribution of *alpha.water* over a cross-section at the half coordinates of the computational domain at time *t=0.05* for `cAlpha` = 0, and at time *t=0.1* when `cAlpha = 1 and 2`

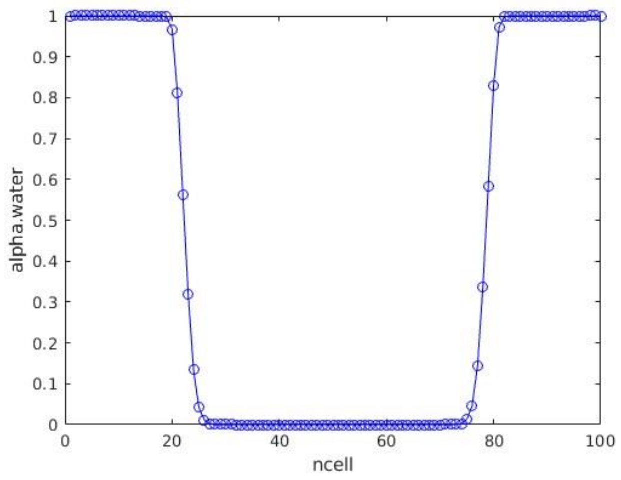


Fig. 13. Evolution of the VOF index function (α) over a cross section, indicating the interface sharpening when `cAlpha=0`

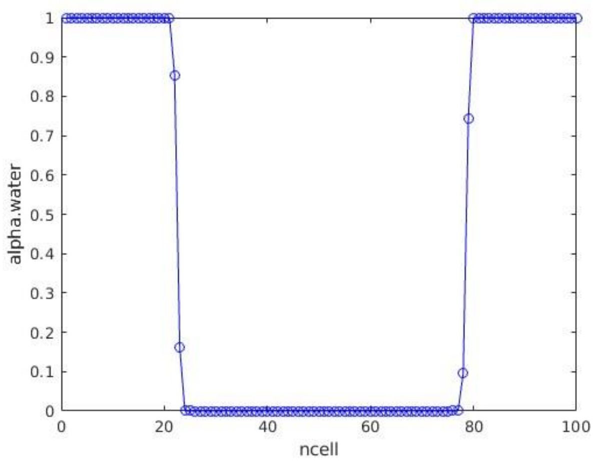


Fig. 14. Evolution of the VOF index function (α) over a cross section, indicating the interface sharpening when `cAlpha=1`

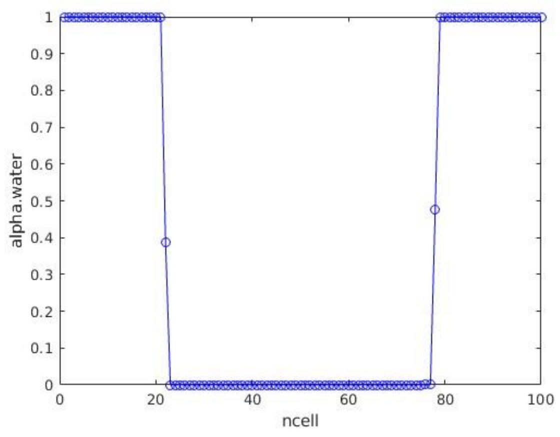


Fig. 15. Evolution of the VOF index function (α) over a cross section, indicating the interface sharpening when `cAlpha=2`

9- We compute the magnitude of the velocity at every single cell by typing the command:

```
postProcess -func "mag(U)"
```

This generates a file named *mag(U)* in each saved time step directory.

10- In order to obtain the maximum magnitude velocity at each time step, we can type:

```
postProcess -func "cellMax(mag(U))"
```

Results are stored in a file named *volFieldValue*, located in the `postProcessing` directory.

11- We plot the maximum magnitude velocity vs. time, for the three cases.

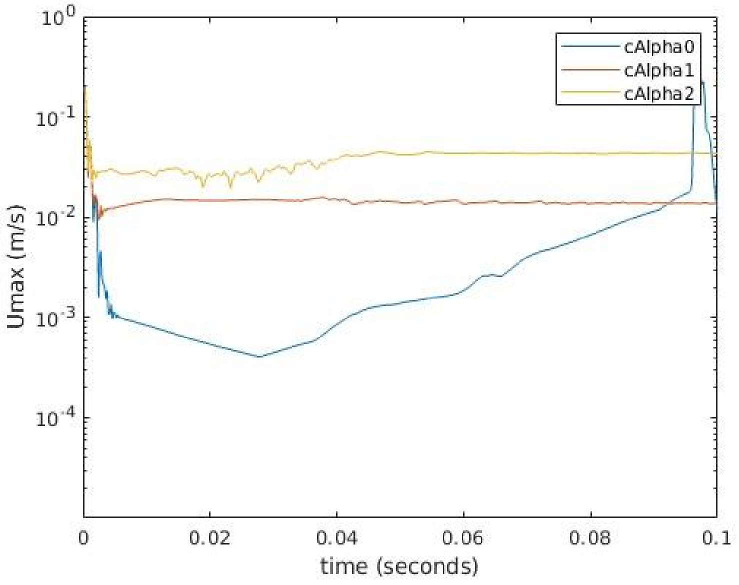


Fig. 16. Evolution of the maximum magnitude velocity through the computational domain, for `cAlpha=0, 1 and 2`

Results: From the figures shown above, it is evident that due to compressing the interface by increasing `cAlpha` from 0 to 1, the corresponding thickness of the interface decreases approximately from 7 grid cells to 4 grid cells. Further increasing `cAlpha` from 1 to 2 not considerably decreases the interface thickness. On the other side, the magnitude of spurious currents significantly increases as `cAlpha` increases. Therefore, we decide to use `cAlpha` = 1 for all the remaining simulations which gives a sharp fluid interface, but at the same time keeps spurious currents relatively small.

12- Compiling the smoother code in OpenFOAM:

- *kva_interfaceProperties* library is the library that includes the smoothing application of interest.

- Starting with Compiling *kva_interfaceProperties* library, whose the whole code documentation is available at https://github.com/floquation/OF-kva_interfaceProperties and linking it to OpenFOAM v8.

- Then, compiling a user defined version of interFoam, linked to *kva_interfaceProperties* library, called myInterFoam_smoothing.

- Renaming the recompiled solver (myInterFoam_smoothing) to distinguish between it and the original one, and thus avoid modifying the existing interFoam solver.

13- The steps from (1) to (8) are repeated for running a new case where `application` in the `controlDict` dictionary file (located in the `system` directory) is modified from `interFoam` (the original VOF solver) to `myInterFoam_smoothing` (the recompiled solver).

14- We plot, on the same graph, the maximum magnitude velocity vs. time for both cases:

   a. no smoothing *(*`cAlpha=1` and *m=0*) and
   b. smoothing (`cAlpha=`*1*, *m=2*)

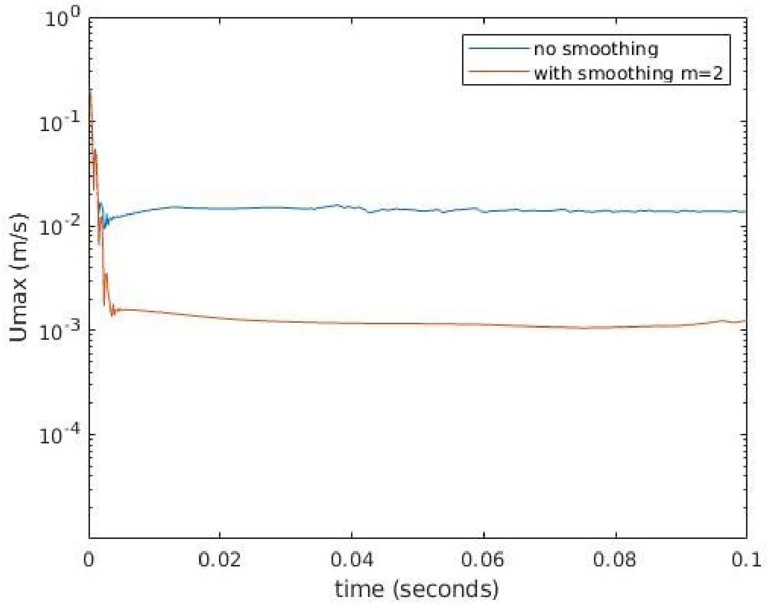


Fig. 17. Evolution of the maximum magnitude velocity through the computational domain, without (*m=0*) and with smoothing (*m=2*)

Results: The graph above evidently shows that the magnitude of spurious currents has been decreased by almost one order of magnitude when applying that Laplacian filter twice (*m=2*)

15- Calculating the exact (analytical) solution of the Laplace pressure through the following formula:

Results: $p_{exact} = \frac{\gamma}{R} = \frac{23.6\times10^{-3}}{169.26\times10^{-6}} = 139.43$ Pa

16- We plot the Laplace pressure magnitude over a cross-section at the half coordinates of the computational domain at the latest time, $t = 0.1$, for both cases:

a. no smoothing *(cAlpha=1* and *m=0*) and
b. smoothing (*cAlpha=1*, *m=2*)

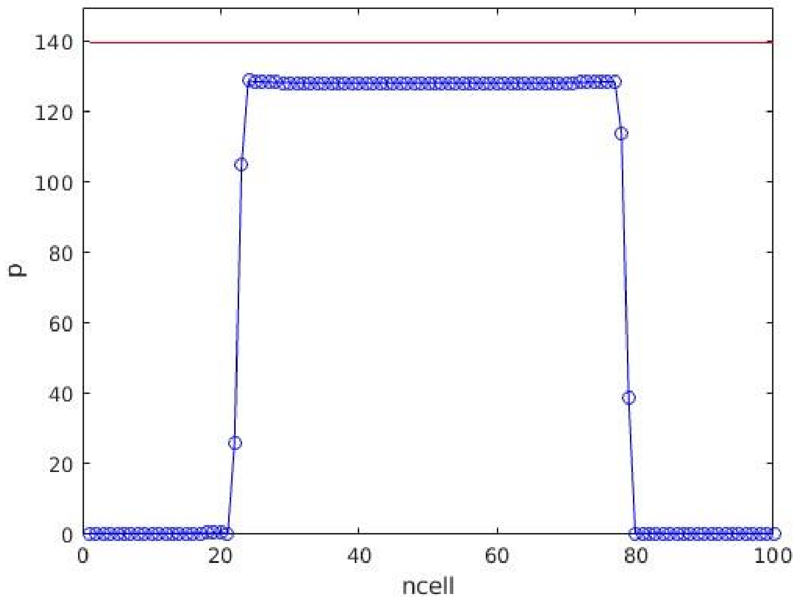


Fig. 18. Computation of the Laplace pressure over a cross section in case of no smoothing (`cAlpha=1`)

Results: The numerical solution of Laplace pressure gives $p_{simulation} = 128.233$ Pa

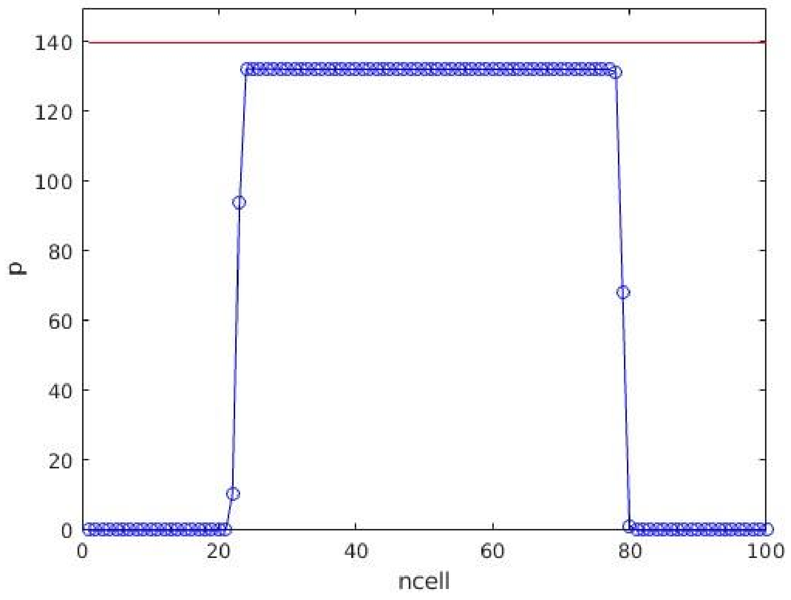


Fig. 19. Computation of the Laplace pressure over a cross section in case of smoothing (`cAlpha=1`)

Results: The numerical solution of Laplace pressure gives $p_{simulation} = 132.243$ Pa

17- We perform an error analysis to evaluate the accuracy of the numerical solution of the Laplace pressure with respect to the exact (analytical) solution.

Results: Error in the Laplace pressure calculation in both cases:

a. no smoothing *(*`cAlpha`*=1* and *m=0*)

$$\%E(p) = \frac{p_{exact} - p_{simulation}}{p_{exact}} = \frac{139.43 - 128.233}{139.43} = 8.0377\%$$

b. smoothing (`cAlpha`*=1*, *m=2*)

$$\%E(p) = \frac{p_{exact} - p_{simulation}}{p_{exact}} = \frac{139.43 - 132.243}{139.43} = 5.1546\%$$

The **error** in the Laplace pressure calculations has decreased by:

$= \frac{8.0377 - 5.1546}{8.0377} = 35.8697\% \approx 36\%$

Chapter 5: Conclusion

Errors in computing the surface tension are mainly induced because of the abrupt change of the VOF index function over the thin interface region between two fluids. These errors lead to non-physical spurious currents with velocities which can be even larger than the actual ones throughout the flow domain. Although several attempts have been made to understand and solve this problem, it is still a main obstacle in the numerical modelling of multiphase flow in porous media. In this work, we have presented a review of the VOF standard solver in OpenFOAM, interFoam, forming one rigorous benchmark: static droplet. We have shown that introducing the interface compression up to the level C_α=1 provides balance between the interface sharpening and the spurious currents magnitude and thus leads to a sharp (thinner) interface and quite high spurious currents. In order to reduce spurious currents, we have smoothed the VOF index function twice using "Laplacian filter", which is not implemented in OpenFOAM library, but we recompiled it from an external code. We have found out that applying the smoother has decreased spurious currents by almost one order of magnitude.

The obtained numerical solution of the Laplace pressure for the static droplet in case of no smoothing was *128.233 Pa*, while in case of smoothing, it was *132.243 Pa*. Therefore, when we performed an error analysis with respect to the exact (analytical) solution of the Laplace pressure for a sphere *(139.43 Pa),* it was proved that the error in the Laplace pressure calculations has decreased by almost 36%.

References

[1] White, Frank (1991), Viscous Fluid Flow. 3rd Edition. McGraw-Hill Mechanical Engineering. ISBN-10: 0072402318.

[2] Stokes, George (1851). "On the Effect of the Internal Friction of Fluids on the Motion of Pendulums". Transactions of the Cambridge Philosophical Society. 9: 8–106.

[3] J.G. Heywood, Viscous Incompressible Fluids: Mathematical Theory, in Encyclopedia of Mathematical Physics, 2006

[4] McLean, Doug (2012). "Continuum Fluid Mechanics and the Navier-Stokes Equations".

[5] Stam, Jos (2003), Real-Time Fluid Dynamics for Games

[6] Kolditz, O. (2002). Finite Volume Method. In: Computational Methods in Environmental Fluid Mechanics. Springer, Berlin, Heidelberg.

[7] Bastian E. Rapp, in Microfluidics: Modelling, Mechanics and Mathematics, 2017

[8] Simon P. Neill, M. Reza Hashemi, in Fundamentals of Ocean Renewable Energy, 2018

[9] Dragan Aleksendrić, Pierpaolo Carlone, in Soft Computing in the Design and Manufacturing of Composite Materials, 2015

[10] Robert Eymard, Thierry Gallouët, Raphaèle Herbin. Finite Volume Methods. J. L. Lions; Philippe Ciarlet. Solution of Equation in n (Part 3), Techniques of Scientific Computing (Part 3), 7, Elsevier, pp.713-1020, 2000, Handbook of Numerical Analysis, 9780444503503. 10.1016/S1570-8659(00)07005-8 . hal-02100732v2

[11] Patankar, Suhas V. (1980). Numerical Heat Transfer and Fluid FLow. Hemisphere Publishing Corporation. ISBN 978-0891165224.

[12] Z. Zhang, Q. Chen, Comparison of the Eulerian and Lagrangian methods for predicting particle transport in enclosed spaces, 2007

[13] Duong A. Hoang, Volkert van Steijn, Luis M. Portela, Michiel T. Kreutzer, Chris R. Kleijn, Benchmark numerical simulations of

segmented two-phase flows in microchannels using the Volume of Fluid method, 2013

[14] Brackbill JU, Kothe DB, Zemach C. A continuum method for modeling surface tension. J Comput Phys 1992;100:335–54

[15] Ali Q. Raeini, Martin J. Blunt, Branko Bijeljic, Modelling two-phase flow in porous media at the pore scale using the volume-of-fluid method. 2012

[16] Venkatesh Inguva, Andreas Schulz and Eugeny Y. Kenig, On methods to reduce spurious currents within VOF solver frameworks. Part 1: a review of the static bubble/droplet, 2020

[17] Kurian J. Vachaparambil and Kristian Etienne Einarsrud, Comparison of Surface Tension Models for the Volume of Fluid Method, 2019

[18] S. Friedlander, Stability of Flows, Encyclopedia of Mathematical Physics, 2006

[19] Francois, Marianne, "A Study of the Volume of Fluid Method for Moving Boundary Problems", master thesis, Embry-Riddle Aeronautical University, Daytona Beach, Florida, 1998

[20] J. Haider, “Numerical Modelling of Evaporation and Condensation Phenomena”, Masters dissertation, Universität Stuttgart, 2013.

[21] Long Chen, Lectures of class “MATH 226A: Computational PDEs”, 2009, University of California, Irvine (UCI).

[22] C.C. Mei, Notes on Advanced Environmental Fluid Mechanics, 2001

[23] C. W. Hirt and B. D. Nichols, Volume of Fluid (VOF) Method for the Dynamics of Free Boundaries, Los Alamos Scientific Laboratory, Los Alamos, New Mexico 87545 Received November 1, 1979

[24] Cyprien Soulaine, 4th Cargèse summer school on flow and transport in porous and fractured media: Introduction to open-source computational fluid dynamics using OpenFOAM® technology, 2018

[25] https://www.grc.nasa.gov/www/k-12/rocket/nseqs.html

[26] https://www.claymath.org

Appendix

Stationary droplet test

1. Directory: `0`

1.1. Dictionary: `alpha.water`

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           volScalarField;
    object          alpha.water;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * *//

dimensions          [0 0 0 0 0 0 0];

internalField       uniform 1;

boundaryField
{

    wall
    {
        type        constantAlphaContactAngle;
        theta0      90;
        limit       gradient;
        value       uniform 1;
    }

    frontAndBack
    {
        type        empty;
    }

}

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

1.2. Dictionary: `P_rgh`

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           volScalarField;
    object          p_rgh;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * *//

dimensions          [1 -1 -2 0 0 0 0];

internalField       uniform 0;


boundaryField
{
    wall
    {
        Type        fixedFluxPressure;
    }

    frontAndBack
    {
        Type        empty;
    }

}

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

1.3. Dictionary: U

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           volVectorField;
    object          U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * *//

dimensions          [0 1 -1 0 0 0 0];

internalField       uniform (0 0 0);

boundaryField
{
    wall
    {
        Type        noSlip;
    }

    frontAndBack
    {
        Type        empty;
    }

}

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

2. **Directory:** `constant`

2.1. Dictionary: `g`

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           uniformDimensionedVectorField;
    location        "constant";
    object          g;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * *//

dimensions          [0 1 -2 0 0 0 0];
value               (0 0 0);


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

2.2. Dictionary: `transportProperties`

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    location        "constant";
    object          transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * *//

phases (water oil);

water
{
    transportModel       Newtonian;
    nu              nu [0 2 -1 0 0 0 0]      1e-06;
    rho             rho [1 -3 0 0 0 0 0]     1000;
}

oil
{
    transportModel       Newtonian;
    nu              nu [0 2 -1 0 0 0 0]      5e-06;
    rho             rho [1 -3 0 0 0 0 0]     500;
}

sigma                    0.0236;

// For smoothing, the following lines are added:

curvatureModel      vofsmooth; // normal;

vofsmoothCoeffs

{

    numSmoothingIterations 2; // default: 2

}
```

2.3. **Dictionary**: `turbulenceProperties`

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    location        "constant";
    object          turbulenceProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * *  //

simulationType      laminar;


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

3. Directory: system

3.1. Dictionary: blockMeshDict

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * *//

convertToMeters 1e-6;

vertices
(
   (0 0 0)
   (600 0 0)
   (600 600 0)
   (0 600 0)
   (0 0 0.01)
   (600 0 0.01)
   (600 600 0.01)
   (0 600 0.01)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (100 100 1) simpleGrading (1
1 1)
);

edges
(
);

boundary
(
     wall
     {
        type   wall;
        faces
        (
          (3 7 6 2)
```

```
              (0 4 5 1)
              (3 7 4 0)
              (2 6 5 1)
          );
      }

              frontAndBack
      {
          type  empty;
          faces
          (
              (0 1 2 3)
              (4 5 6 7)
          );
      }

);

mergePatchPairs
(
);
```

3.2. Dictionary: controlDict

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    location        "system";
    object          controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * *//

application         interFoam;      //myInterFoam_smoothing

startFrom           latestTime;

startTime           0;

stopAt              endTime;//writeNow;//

endTime             0.1;

deltaT              1e-7;

writeControl        adjustableRunTime;

writeInterval       2e-4;

purgeWrite          0;

writeFormat         ascii;

writePrecision      8;

writeCompression    off;

timeFormat          general;

timePrecision       6;

runTimeModifiable   yes;

adjustTimeStep      yes;

maxCo               0.3;
```

```
maxAlphaCo          0.3;

maxDeltaT           9e-7;


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

3.3. **Dictionary**: `decomposeParDict`

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    location        "system";
    object          decomposeParDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * *//

numberOfSubdomains  4;

method              scotch;




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

3.4. Dictionary: fvSchemes

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    location        "system";
    object          fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * *//

ddtSchemes
{
    default              Euler;
}

gradSchemes
{
    default              pointCellsLeastSquares;
    grad(U)              Gauss linear;
}

divSchemes
{
    div(rhoPhi,U)  Gauss linearUpwind grad(U);
    div(phi,alpha)  Gauss vanLeer;
    div(phirb,alpha) Gauss interfaceCompression;
    div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;
}

laplacianSchemes
{
    default              Gauss linear corrected;
}

interpolationSchemes
{
    default              linear;
}

snGradSchemes
{
    default              corrected;
```

3.5. Dictionary: `fvSolution`

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    location        "system";
    object          fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * *//

solvers
{
    "alpha.water.*"
    {
        nAlphaCorr       1;
        nAlphaSubCycles  2;
        cAlpha           0;   //1; 2;

        MULESCorr        yes;
        nLimiterIter     5;

        solver           smoothSolver;
        smoother         symGaussSeidel;
        tolerance        1e-8;
        relTol           0;
    }

    "pcorr.*"
    {
        solver           PCG;
        preconditioner   DIC;
        tolerance        1e-10;
        relTol           0;
    }

    p_rgh
    {
        solver           PCG;
        preconditioner   DIC;
        tolerance        1e-07;
        relTol           0.05;
    }
    p_rghFinal
    {
        $p_rgh;
        relTol                0;
    }

    U
    {
        solver           smoothSolver;
        smoother         symGaussSeidel;
        tolerance        1e-06;
        relTol           0;
    }
}

PIMPLE
{
    momentumPredictor         no;
    nOuterCorrectors          1;
    nCorrectors               3;
    nNonOrthogonalCorrectors  0;
    pRefCell                  1;
    pRefValue                 0;
}

relaxationFactors
{
    equations
    {
        ".*" 1;
    }
}


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

3.6. Dictionary: `setFieldsDict`

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    location        "system";
    object          setFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * *//

defaultFieldValues
(
    volScalarFieldValue alpha.water 1
);

regions
(
    boxToCell
    {
        box (150e-6 150e-6 0) (450e-6 450e-6 0.01);
        fieldValues
        (
            volScalarFieldValue alpha.water  0
        );
    }
);


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