Mario Anton Schriefl, BSc

Implementation of a Quadrature Method Of Moments in OpenFOAM to Study Condensation on Sub-Micron Particles

MASTER THESIS

for obtaining the academic degree
Diplom-Ingenieur

Master Programme of Technical Physics

Graz University of Technology

Supervisor:
Ao. Univ.-Prof. Dipl.-Ing. Dr.techn. Martin Heyn
Institute of Theoretical and Computational Physics

Co-Supervisor:
Ass.Prof. Dipl.-Ing. Dr.techn. Stefan Radl
Institute of Process and Particle Engineering

Graz, November 2014

EIDESSTATTLICHE ERKLÄRUNG

AFFIDAVIT

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Datum / Date	Unterschrift / Signature

Abstract

Condensation particle counters (CPCs) are used to measure the number concentration of sub-micron particles in the exhaust gas of combustion engines. CPCs rely on the condensation of vaporized working fluid onto the particles, and the subsequent formation of sufficiently large droplets, that can be easily detected. In order to optimize the performance of CPCs, a precise understanding of the physical processes affecting droplet nucleation and growth is of central importance.
To model the formation and subsequent growth of droplets in the evaporator-condenser system of a CPC, a computer program, specifically the library *'qmomCloud'*, was developed and implemented into the open-source software package OpenFOAM. *'qmomCloud'* is an extension of the existing *cpcFoamCompressible* solver, which was developed to predict the vapor concentration profiles inside a CPC. The library *'qmomCloud'* has routines to solve the population balance equation (PBE) of the droplets using the quadrature method of moments (QMOM). Also, the local depletion of the vapor concentration due to the formation and growth of the droplets, as well as the effect of the heat of condensation can be modeled. The *qmomCloud* library allows to take into account these effects by calculating the respective source terms in the transport equations for heat and vapor concentration in the exhaust gas. Furthermore, the *qmomCloud* library offers a general framework for the implementation of additional physical models (e.g., coagulation). Thus, the library provides an environment for fast and efficient prediction of the size distribution of polydisperse aerosols, and can be used to study a variety of applications in the field of CPC development.
The reliability and stability of the numerical algorithms implemented in the *qmomCloud* library, as well as the implemented physical models for nucleation and growth were rigorously tested. The library was employed to study the condensation phenomena in an evaporator-condenser system of a CPC, including a detailed assessment of coupling effects, i.e., effects due to latent heat release and the depletion of the vapor in the exhaust gas. It was found that the droplets grow to a final size of approximately 19 μm without coupling, and that coupling effects lead to a final size of approximately 14 μm.

Acknowledgement

In the course of writing this thesis I received great support from many people, for which I am very grateful.

First I would like to thank Ass.Prof. Dipl.-Ing. Dr.techn. Stefan Radl for his brilliant mentoring and inspiring guidance throughout the entire process. I would also like to thank Ao. Univ.-Prof. Dipl.-Ing. Dr.techn. Martin Heyn for his constructive supervision. Furthermore I would like to thank Dr. Athanasios Mamakos and Dr. Verena Vescoli of AVL List GmbH for giving me the possibility to write this thesis and for their great continuous support. Moreover, I would like to express my gratitude to Dipl.-Ing. Tristan Reinisch, who was always available whenever problems occurred.

Additionally, I am taking this opportunity to express my gratitude to my family: I wish to thank my parents Alfred and Gertrude for their financial and mental support over the last few years; my brother Christoph for encouraging me when things did not go well; and my girlfriend Anna-Maria, for her love and care, helping me to maintain my emotional equilibrium throughout the course of my studies.

Contents

List of Figures

List of Tables

Nomenclature

$\boldsymbol{A_p}$	particle acceleration (continuous rate of change of particle velocity)	$\left[\frac{m}{s^2}\right]$
A_p	particle surface area	$[m^3]$
a_α, b_α	recursion coefficients for the polynomials P_α	
B_m	Spalding mass transfer number	$[-]$
$\boldsymbol{b}_k$	response vector in the McGraw correction algorithm	
c_k	correction factor in the McGraw correction algorithm	
c_p	specific heat	$\left[\frac{J}{K}\right]$
D_v	vapor molecular diffusion coefficient	$\left[\frac{m^2}{s}\right]$
D_ξ	phase-space diffusion coefficient	
$d_{d,init}$	initial droplet diameter	$[m]$
d_k	Kelvin diameter (critical nucleation diameter)	$[m]$
d_{mol}	molecular diameter	$[m]$
$\boldsymbol{d_n}$	n^{th} difference vector used in the McGraw correction method	
d_p	primary particle diameter	$[m]$
$\overline{d}$	moment limitation factor	
f_g	multiplicative factor for the affinity of particles for the working fluid in heterogeneous nucleation theory	$[-]$
G	droplet growth rate	$\left[\frac{m}{s}\right]$
G_0	power law growth rate	$\left[\frac{m}{s}\right]$
$\dot{H}_{source}$	coupling source for heat transfer	$\left[\frac{J}{m^3 s}\right]$
h	specific enthalpy of the continuous phase	$\left[\frac{J}{kg}\right]$
h_v	specific enthalpy of vaporization (or condensation)	$\left[\frac{J}{kg}\right]$
J	nucleation rate	$\left[\frac{1}{m^3 \cdot s}\right]$
$\boldsymbol{J}$	Jacobi matrix used in the PD algorithm	
J_{het}	heterogeneous nucleation rate	$\left[\frac{1}{m^3 \cdot s}\right]$
J_0	kinetic prefactor of the nucleation rate	$\left[\frac{1}{m^3 \cdot s}\right]$
J^0_{het}	kinetic prefactor of the heterogeneous nucleation rate	$\left[\frac{1}{m^3 \cdot s}\right]$
j_{cond}	condensation flux	$\left[\frac{mol}{s \cdot m^2}\right]$
L_{10}	arithmetic mean particle size	$[m]$
L_{32}	Sauter mean diameter	$[m]$
$L_{k+1,k}$	average mean particle size	$[m]$
K	geometric shape factor	$[-]$
k_B	Boltzmann constant	$\left[\frac{kg \cdot m^2}{s^2 \cdot K}\right]$
k^*	index of the changed moment within the McGraw algorithm	
m_k	k^{th} moment of the NDF with respect to the internal coordinate(s), sometimes called scalars	$[m^{k-3}]$

m_k^{exp}	expected value of the k^{th} moment within the moment limitation	$\left[m^{k-3}\right]$
m_p	particle mass	$[kg]$
$\dot{m}_p$	rate of mass transfer to the disperse phase	$\left[\frac{kg}{s}\right]$
MW_v	molecular weight of the vapor	$\left[\frac{kg}{mol}\right]$
N_p	total particle number concentration	$\left[\frac{1}{m^3}\right]$
N_A	Avogadro constant	$\left[\frac{1}{mol}\right]$
N_{prim}	primary concentration of seed particles	$\left[\frac{1}{m^3}\right]$
n	number density function (NDF), (the unit is valid for particle length as internal coordinate)	$\left[\frac{1}{m^4}\right]$
P_{act}	activation probability for heterogeneous nucleation	$[-]$
$\boldsymbol{P}$	matrix for the PD algorithm	
$P_N(\xi)$	polynomial of order N that is orthogonal with respect to a weight function $n(\xi)$	
p	pressure	$\left[\frac{N}{m^2}\right]$
p_v	vapor pressure of the working fluid	$\left[\frac{N}{m^2}\right]$
R	coordinate along the radial pipe axis	$[m]$
R_W	wall radius of the pipe	$[m]$
r	exponent of the power law growth rate	
r^*	critical radius of homogeneously nucleated cluster	$[m]$
S	supersaturation (also: saturation ratio)	$[-]$
$\mathcal{S}_1$	discontinuous source term representing discrete events	
$\dot{\mathcal{S}}_{coupl}$	source term in the mass transport equation	$\left[\frac{kg}{m^3\,s}\right]$
$\dot{\mathcal{S}}_{heat}$	source term in the heat transfer equation	$\left[\frac{J\,kg}{m^3 s}\right]$
Sh	Sherwood number	$[-]$
T_f	working fluid temperature	$[K]$
T_c	critical temperature of the working fluid	$[K]$
t	time	$[s]$
V_p	particle volume	$[m^3]$
$\dot{V}$	volumetric flow rate	$\left[\frac{m^3}{s}\right]$
$\boldsymbol{v}$	particle velocity	$\left[\frac{m}{s}\right]$
$\boldsymbol{u}$	fluid velocity	$\left[\frac{m}{s}\right]$
u_{av}	average velocity of the laminar flow within the pipe	$\left[\frac{m}{s}\right]$
w_α	weights of the Gaussian quadrature approximation	
X	coordinate along the cylindrical pipe axis	$[m]$
X'	fraction of seed particle size and Kelvin diameter	$[-]$
$\boldsymbol{x}$	spatial (external) coordinates (= position of particles in phase space)	$[m]$
Y	normalized vapor mass fraction	$[-]$
Y_{f1}	mass fraction of evaporating molecules in the bulk	$[-]$

Y_{s1}	mass fraction of evaporating molecules near the interface	$[-]$
y_{sat}	mass fraction of saturated vapor	$[-]$
Γ	(particle) diffusion coefficient for diffusion in physical space	$\left[\frac{m^2}{s}\right]$
ΔG^*_{hom}	free energy of formation of a homogeneously nucleated critical cluster	$\left[\frac{kg \cdot m^2}{s^2}\right]$
Δh	change in the specific enthalpy	$\left[\frac{J}{kg}\right]$
Δt	time step	$[s]$
ΔV	physical space volume element	$[m^3]$
Δx	physical space grid spacing	$[m]$
δ_χ	kernel density function used in EQMOM	
ζ_α	coefficient of the continued fraction in the PD algorithm	
θ	three-phase contact angle	$[-]$
λ	heat conductivity	$\left[\frac{W}{mK}\right]$
μ'	parameter of the Gaussian (mean value) or log-normal distribution function	
μ_f	fluid viscosity	$\left[\frac{kg}{m \cdot s}\right]$
$\tilde{\nu}$	solute molecular volume	$[m^3]$
$\boldsymbol{\xi}$	property vector $\xi \equiv (\xi_1, \xi_2, ..., \xi_M)$ containing M internal coordinates	
ξ	droplet or particle size (primary internal coordinate)	$[m]$
$\dot{\boldsymbol{\xi}}$	rate of change of internal coordinate(s)	
ξ_α	nodes of the Gaussian quadrature approximation	
ρ	total gas density	$\left[\frac{kg}{m^3}\right]$
ρ_v	(partial) density of the vaporized phase (working fluid)	$\left[\frac{kg}{m^3}\right]$
ρ_l	liquid density of the droplets (i.e., condensed working fluid)	$\left[\frac{kg}{m^3}\right]$
$\rho_{v,sat}$	concentration of the saturated vapor	$\left[\frac{kg}{m^3}\right]$
$\rho^{ref}_{v,sat}$	reference concentration of the saturated vapor in the evaporator	$\left[\frac{kg}{m^3}\right]$
σ	surface tension between condensed vapor and fluid (i.e., gas)	$\left[\frac{kg}{s^2}\right]$
σ'	parameter of the Gaussian (variance) or log-normal distribution function	
χ	parameter used to define the kernel density function in EQMOM	
$\Omega_{\boldsymbol{x}}$	control volume in physical space	$[m^3]$
Ω_ξ	control volume of phase space (the unit is valid for particle length as internal coordinate)	$[m]$

Abbreviations

APC	AVL particle counter
CFD	Computitional fluid dynamics
CPC	Condensation particle counter
DQMOM	Direct quadrature method of moments
EQMOM	Extended quadrature method of moments
FVM	Finite volume method
GPBE	Generalized population balance equation
ML	Matlab®
MOM	Method of moments
MOMIC	Method of moments with interpolative closure
OF	OpenFOAM®
PBE	Population balance equation
PD	Product-difference
QMOM	Quadrature based method of moments

1 Introduction

1.1 Motivation

The measurement of the particle number concentration in the exhaust gas of combustion engines has recently formed an integral part of emission standards. For instance, the European Union has restricted the emission of particles by newly registered diesel passenger cars to $6 \cdot 10^{11}$ particles per km [1].
The most common measurement technique for counting the emitted solid particles is represented by a condensation particle counter (CPC). In a CPC, the exhaust gas is passed through a device that is able to generate a certain supersaturation level by subsequent evaporation and condensation of a working fluid (e.g., n-Butanol). In the condenser the working fluid condenses onto the particles. Afterwards, the aerosols grow to a sufficiently large size (in the range of several microns) which allows the detection of them using a light scattering technique [2]. Because the volumetric flow rate of the exhaust gas is known, the particle concentration and size distribution can be determined by simply counting the number of particles with a certain size.

With the AVL Particle Counter (APC) AVL List GmbH provides a widely used tool to measure particle number concentration in accordance with the particle measurement programme of the United Nations Economic Commission for Europe [3], [4]. The APC uses a TSI 3790 Condensation Particle Counter as core sensor. In recent studies it was found out that the performance of the TSI 3790 CPC varies for different aerosols. Furthermore, it turned out that the counting efficiency decreases with increasing operation time. This degradation of the CPC performance is a direct indication of droplet size reduction [5]. Hence, a precise understanding of the physical processes affecting droplet growth is key for the further optimization of CPCs that do not suffer these limitations.

To model droplet growth inside a CPC, the population balance equation (PBE) of droplets must be solved in conjunction with the governing equations for mass, heat, and species conservation [6]. The PBE could either be simulated using a Lagrangian approach, where each particle is tracked (i.e., a direct approach), or the PBE can be solved in an Eulerian frame of reference. For the latter approach, often transport equations for a set of lower-order moments of the particle size distribution are solved. A popular method is the Quadrature Method of Moments (QMOM), which computes missing moments, not included in the moment set solved for directly, using Gaussian quadrature. The QMOM has the advantage that, compared to a Lagrangian approach, a very much lower number of equations has to be solved, resulting in a low computation time [7].

1.2 Goals

The overall goal of this thesis is to develop a simulation tool in order to understand the droplet grow process inside a CPC. Specifically, the achievable droplet size distribution should be computed, and appropriate operation conditions (e.g., the temperatures of the condenser and the evaporator) for a successful operation of a CPC should be defined. The simulation tool should be based on an already existing CFD solver *cpcFoamCompressible* that was developed by Stefan Radl (TU Graz) and Tristan Reinisch (AVL List GmbH). The new tool (named *qmomCloud* in the following) should use the QMOM approach to compute the local droplet size distribution, and should be implemented as a library that can be linked to any OpenFOAM® application, e.g., *cpcFoamCompressible*.

Specifically, the *qmomCloud* library should model the condensation process (i.e., nucleation and growth of droplets) of the working fluid onto primary particles present in the exhaust gas. Since the formation and growth of droplets causes a local depletion of the working fluid's concentration, as well as a release of condensation heat, the exhaust gas is affected by condensation processes. This coupling should be modeled as well in order to better understand local vapor concentration and temperature. Furthermore, convective and diffusive transport of the particles must be modeled, since (i) the gas velocity profiles within a CPC is not uniform, (ii) and sub-micron particles can be expected to diffuse with appreciable rates.
Within this thesis, a general framework for the implementation of physical models was established such that nucleation, growth, as well as convective and diffusive transport of the particles can be modeled. The framework is such that the *qmomCloud* library can be extended easily to model other physical models, e.g., coalescence or breakage. In summary the library provides an environment for fast and efficient prediction of the size distribution of polydisperse aerosols, suitable to study a variety of applications in the field of CPC development.

1.3 Thesis Outline

After a literature research regarding QMOM (see the recent book of [8]), the core numerical algorithms of the QMOM were implemented and tested in Matlab®. Subsequently, all required physical models were documented, implemented in Matlab and tested to provide reference results for *qmomCloud*.
The next step was the implementation of the *qmomCloud* library in the OpenFOAM environment. The reliability and functionality of the solver was investigated in a simple test geometry. Finally, the solver was applied to a pipe flow simulation case that simulates the flow in the evaporator-condenser system of a CPC. Effects due to the depletion of the working fluid vapor in the exhaust gas were studied.

Chapter 2 provides a short description of the existing *cpcFoamCompressible* solver and how this code can be used to predict gas flow, as well as local vapor concentration and temperature inside a CPC.
In Chapter 3 the theoretical backgrounds for solving a population balance equation for describing the size distribution of a particle population are documented. Furthermore, the mathematical basics for QMOM, as well as a detailed description of the physical models used in this work, are presented.
Chapter 4 details the software architecture of the *qmomCloud* library, and how *qmom-Cloud* can be employed for solving a particular flow problem. Furthermore, a short introduction to the underlying software package, i.e., OpenFOAM, is provided.
In Chapter 5 the correct implementation of the *qmomCloud* library, and the coupling to the *cpcFoamCompressible* solver was examined. Numerical details connected to the use of the QMOM, as well as the reliability of the used physical models were studied and results are summarized in this section.
Chapter 6 provides a description of the existing pipe flow simulation case, key settings used to control the *qmomCloud* library, and key results.
Finally, in Chapter 7 the key outcome of the thesis is summarized. An outlook with respect to further improvements and applications of the *qmomCloud* library, as well as the *cpcFoamCompressible* solver is provided.

2 Previous CPC Simulation Studies

2.1 CPC Working Principle

With a CPC the number concentration of sub-micron particles (e.g., soot) within a sampled exhaust gas flow cannot be measured directly, since direct particle detection is very difficult (e.g., the intensity of light scattered by sub-micron is very low). Consequently, the particles are used as seeds for the condensation of a vapor, i.e., droplets are formed, which then can be detected using standard light scattering methods. Specifically, a working fluid is condensed onto the particles, i.e., heterogeneous nucleation and growth takes place. This is done in the evaporator-condenser system of the CPC with moderate supersaturation levels to avoid homogeneous nucleation.
In the evaporator the working fluid is evaporated. In the condenser the vaporized working fluid becomes supersaturated in the exhaust gas, and thus, the conditions for nucleation of droplets onto the particles are achieved. Once droplets have been formed, they grow to a size in the range of a few microns.

2.2 Single-Phase Studies

Numerous studies have been performed to study single-phase flow in CPCs with the aim to predict local supersaturation levels by means of an analytical or numerical approach. An analytical solution for the supersaturation profile in the condenser section of the pipe can be obtained by solving a boundary value problem in a cylindrical geometry. This analytical approach, known as the Graetz Problem [9], is presented in [10]. The work is based on the numerical model developed by Giechaskiel [11]. However, this solution is only valid for constant transport properties and gas density, as well as uniform boundary conditions. The latter is especially problematic for the vapor concentration at the inlet of the condenser section, since full saturation of the exhaust gas may not be achieved in the evaporator section. Consequently, a more general numerical approach via computational fluid dynamics (CFD) has been adopted in more recent studies.

The *cpcFoamCompressilbe* solver is a CFD code based on OpenFOAM$^{®}$ that was developed by Stefan Radl (TU Graz) and Tristan Reinisch (AVL List GmbH) to investigate the behavior of the working fluid vapor in the evaporator-condenser system of a CPC. The solver allows to (i) model heat and species transport in a compressible fluid (i.e., the exhaust gas), (ii) accounts for changes in the gas density due to local pressure and temperature (e.g., using the ideal gas equations of state), and (iii) accounts for temperature dependent transport properties (e.g., diffusion coefficients).

The working fluid inside the evaporator-condenser system of a CPC can be modeled by employing the *cpcFoamCompressilbe* solver to the existing pipe flow simulation case. Inducing a (laminar) fluid flow in the pipe, and different wall temperatures for the evaporator section (i.e., the high temperature region) and the condenser section (i.e., the low temperature region) of the pipe (see Fig. 6.1), a steady state solution is obtained. The result of a simulation is the profile of the supersaturation level inside the pipe that indicates regions where heterogeneous nucleation of the working fluid onto seed particles can take place.

2.3 Droplet Growth Studies

The growth of droplets in the condenser is described in [10] and [12] where the growth model of Fuchs [13] was used. In this work, for the growth studies the operating conditions of a TSI 3790 CPC were employed. The outcome was that for seed particles in the range from 17.7 to 100 nm the droplets grow up to a final size (at the outlet of the condenser) in the range of 5 to 8 μm. In this previous work, however, the growth was decoupled from the flow problem.
The same applies to the work of Ahn [14] where the same growth model [13] was used. Within this work the droplet growth in a TSI 3020 CPC was modeled with the result that 20 nm nuclei grow to a size of about 12.3 μm. In both studies the sufficient droplet size of about 0.5 μm [14] for the detection with light scattering was exceeded.
Furthermore, in [10] the depletion of the working fluid vapor concentration was estimated by reducing the mass of the vapor by the mass of the droplets that have grown to their final size (worst case scenario). Note, these calculations were decoupled from the mass and heat transfer problem. In case of 40 nm seed particles a maximum reduction in the saturation ratio of about 3% was found.

3 Theoretical Backgrounds

3.1 Modeling of Multiphase Flows

Disperse multiphase flows consist of a disperse phase and a continuous phase. Their understanding in terms of a quantitative model is essential to describe, for example, combustion processes, particle size measurement devices, or particle transport in the atmosphere. Computational models for disperse multiphase flows can be categorized in (i) models describing the evolution of the disperse phase (e.g., particle growth, coagulation, breakage; this can be achieved by solving a population balance equation (PBE)), as well as (ii) models for simulating multiphase flow. The latter models consist of spatially inhomogeneous mass, momentum and energy balances of the disperse and the continuous phase. Typically these models are discretized using the finite-volume method (FVM) in the context of computational fluid dynamics (CFD).

This Chapter of the thesis is focused on the first type of models (i.e., models for describing the evolution of the disperse phase), and thus, it will provide an overview of the underlying governing equations. Specifically, we will start with a general introduction to the concepts of a PBE, and then focus on the application to Condensation Particle Counters (CPCs). As the disperse phase in CPCs consists of droplets (that form around small primary particles with a predefined size distribution), the term 'droplets' and 'particles' are interchangeable in what follows. Note, that one could use the concepts explained below to describe the size distribution of the primary particles as well. However, such an analysis is beyond the scope of this text.

3.1.1 Number Density Function

The number density function (NDF) $n(t, \boldsymbol{x}, \boldsymbol{\xi})$ gives the expected number of particles per infinitesimal phase-space volume $d\boldsymbol{\xi}$ and infinitesimal physical space interval $d\boldsymbol{x}$. The NDF depends on time t, external coordinates $\boldsymbol{x}$ and internal coordinates $\boldsymbol{\xi}$, whereas the length of the internal coordinate vector is M ($\boldsymbol{\xi} \equiv (\xi_1, \xi_2, ..., \xi_M)$). We are here concerned with an univariate NDF, for which the length of the property vector is $M = 1$. That means there is only one internal coordinate ξ. Multivariate distributions would have more than one internal coordinate, e.g., a bivariate NDF describing a distribution having the internal coordinates particle length (or diameter) and particle volume would be based on $\xi_1 = d_p$ and $\xi_2 = V_p$. Unfortunately, the numerical description of multivariate distributions is significantly more involved. Because the droplets in a CPC are well characterized by their diameter, we will not discuss these distributions, and focus on the **length-based NDF** distribution $n(t, \boldsymbol{x}, d_p)$, i.e., $\xi = d_p$. In what follows, we simply use n to denote the NDF.

In order to apply the concept of NDFs to describe polydisperse flows, it is useful to discuss which physical information can be extracted from an NDF. For example, the number of particles with properties between ξ and $(\xi + \Delta\xi)$ in a control volume $\Omega_{\boldsymbol{x}}$ is given by

$$\Delta N_p = \Omega_x n \Delta\xi \tag{3.1}$$

Next, the k^{th} moment of the NDF is given by

$$m_k = \int_{\Omega_\xi} \xi^k n d\xi \tag{3.2}$$

The moments of the NDF reflect certain average quantities of the particle population. For example, the 0^{th} moment equals the total particle number concentration, i.e., $m_0 = N_p$. Note that n, compared to a probability density function, is not necessarily normalized. In case it would be the value of m_0 would always be unity.
The (arithmetic) mean particle size is defined as

$$L_{10} = \frac{1}{N_p} \int_0^\infty d_p n d(d_p) = \frac{m_1}{m_0} \tag{3.3}$$

More generally, a set of average diameters can be defined based on the ratio of certain moments

$$L_{k+1,k} = K \frac{m_{k+1}}{m_k} \tag{3.4}$$

for any k, where the choice of k determines the physical meaning of the L. For $k = 2$, and in case the particles are spherical, one gets the **Sauter mean diameter** L_{32}, which represents the diameter of a sphere with the same surface area to volume ratio as the particle population. Note that also other integral properties (e.g., the total surface area) can be computed using the moments and a geometrical prefactor K [15]. In case we assume spherical particles, $K = \pi/6$.

3.1.2 Transport Equations

Considering all possible changes of the NDF in a finite physical space and phase-space control volume, one can write a particle number balance

$$\frac{\partial}{\partial t}\left(\int_{\Omega_{\boldsymbol{x}}} d\boldsymbol{x} \int_{\Omega_\xi} n d\boldsymbol{\xi}\right) + \int_{\Omega_\xi} d\boldsymbol{\xi} \int_{\partial\Omega_{\boldsymbol{x}}} (n\boldsymbol{v}) \cdot d\boldsymbol{A}_{\boldsymbol{x}} + \int_{\Omega_{\boldsymbol{x}}} d\boldsymbol{x} \int_{\partial\Omega_\xi} (n\dot{\boldsymbol{\xi}}) \cdot d\boldsymbol{A}_\xi = \int_{\Omega_{\boldsymbol{x}}} d\boldsymbol{x} \int_{\Omega_\xi} d\boldsymbol{\xi} \mathcal{S}_1 \tag{3.5}$$

The first term on the left hand side of Eq. (3.5) describes the accumulation of particles, and the second and third term describe convection in physical and phase-space, respectively. Physically speaking, these latter terms reflect continuous changes of the (internal and external) coordinates of the particle population. In contrast, the term on the right-hand side of Eq. (3.5) describes discontinuous (or discrete) events, e.g., collision of particles and subsequent coagulation, or nucleation.
Applying the Reynolds-Gauss theorem yields the **population balance equation** (PBE) [8, p. 36]

$$\frac{\partial n}{\partial t} + \frac{\partial}{\partial x_i}(v_i n) + \frac{\partial}{\partial \xi_i}\left(\dot{\xi}_i n\right) = \mathcal{S}_1 \tag{3.6}$$

In case the NDF includes the particle velocity as internal coordinate (i.e. $n(t, \boldsymbol{x}, \boldsymbol{v}, \xi)$), one can derive the **generalized population balance equation** (GPBE) in a similar way.

$$\frac{\partial n}{\partial t} + \frac{\partial}{\partial x_i}(v_i n) + \frac{\partial}{\partial v_i}(A_{p,i} n) + \frac{\partial}{\partial \xi_i}\left(\dot{\xi}_i n\right) = \mathcal{S}_1 \tag{3.7}$$

The third term on the left-hand side of Eq. 3.7 describes the acceleration due to an external force acting on the particles. In the simplest case, in which the velocity is the only internal coordinate, the GPBE is the well known Boltzmann equation [8] .

In case the particles (or droplets) are small, and/or the disperse-to-primary-phase density ratio is small (i.e., the Stokes number is small), the particle velocity quickly adjusts to that of the surrounding fluid. This assumption of equal fluid and particle velocity fields is known as the **dusty-gas model** [8].
If, beside that assumption, the fluid flow is laminar and steady, the GPBE can be greatly simplified. Specifically, the transport due to advection can be computed directly from the fluid's velocity, and diffusive transport (of the particle population) can be modeled with an effective particle diffusion coefficient Γ. The resulting transport equation is then called the **Laminar PBE**

$$\frac{\partial n}{\partial t} + \frac{\partial}{\partial x_i}\left(u_i n - \Gamma \frac{\partial n}{\partial x_i}\right) + \frac{\partial}{\partial \xi_i}\left(\dot{\xi}_i n\right) = \mathcal{S}_1 \tag{3.8}$$

Note that beside diffusion of the particles (or droplets) in the fluid (= physical-space diffusion), also phase-space diffusion could be modeled. Phase-space diffusion would describe e.g. the fluctuation of the growth rate in the case where the fluid condensates onto particles. Because the above assumptions are valid for most CPCs, the laminar PBE is thought to be sufficiently general to describe the particle population in a CPC. Phase-space diffusion is of minor interest and will therefore not be treated. Consequently, we focus only on the transport equation Eq. (3.8) for the rest of the discussion.

Often, the NDF cannot be calculated directly, since the direct solution of, e.g., the laminar PBE, is very demanding. Thus, in many engineering applications one only aims for a solution to some average properties of the NDF, i.e., of its moments. Unfortunately, the terms in the PBE, GPBE, and the laminar PBE require the knowledge of the full NDF. However, to get the full NDF an infinite set of moments is required. Since we can only deal with a set of lower order moments, the full NDF cannot be reproduced exactly. This problem is referred to as **the closure problem** (see the discussion in the next chapter), and hence aims on approximating n from average information. Thus, one needs to derive an equation to describe the evolution of the moments m_k first, and then apply a technique to approximate the NDF. The former can be achieved by integrating the transport equations Eq. (3.6) or Eq. (3.8) over the phase space. The resulting **moment-transport equations** for a length-based NDF with the internal coordinate $\xi = d_p$, and described by the laminar PBE is

$$\int_0^\infty \frac{\partial}{\partial t}(d_p)^k n \cdot d(d_p) + \int_0^\infty \frac{\partial}{\partial x_i}\left(u_i n\right)(d_p)^k \cdot d(d_p) + \int_0^\infty kGn(d_p)^{k-1} \cdot d(d_p) \\ = \int_0^\infty \mathcal{S}_1 (d_p)^k \cdot d(d_p) \qquad (3.9)$$

Here $\dot{\xi} = G$ is a growth rate describing, for example, the increase of the droplet size due to condensation. In the simple situation where G is independent from d_p, we obtain a closed set of moment-transport equations

$$\frac{\partial m_k}{\partial t} + \frac{\partial}{\partial x_i}\left(u_i m_k\right) = kGm_{k-1} + \mathcal{S}_{1,k} \qquad (3.10)$$

The procedure of transforming the PBE to a set of transport equations for the moments of the NDF is called **Method of Moments (MOM)**. The idea of calculating the time evolution of the moments instead of the time evolution of the full NDF has two main advantages:

- Reduction of the dimensionality of the problem by integrating out the internal coordinate
- Moments correspond to measurable quantities like the total particle number density, and can be directly used, e.g., in engineering applications.

Nevertheless, there are also some drawbacks, as the loss of information due to the finite set of moments, as an arbitrary NDF can only be expressed exactly with an infinite set of moments. The other disadvantage is that the source term cannot be written as a function of moments only.

E.g. for the 0^{th} moment, the growth term in Eq. (3.10) is zero, because the number concentration of particles (or droplets) is not affected by condensation. It is now critical to realize, that in case the moment-transport equation for the k^{th} moment involves moments of order higher than k (or other expressions that contain the internal variable ξ), the transport equation for the k^{th} moment is not closed. The next session will provide a more detailed explanation of this problem when attempting to solve the system of moment transport equations.

3.1.3 The Closure Problem

For the solution of the moment transport equations it is essential to know whether the system of moment equations is closed or not. Therefore, in this section the term 'closure problem', that was already mentioned in Sec. 3.1.2, will be defined more closely.

Generally, one wants to solve a system of moment transport equations up to a certain order k_{max} that can be written as

$$\begin{aligned}
\frac{\partial m_0}{\partial t} &= f(m_0, m_1, ..., m_{n_0}) \\
\frac{\partial m_1}{\partial t} &= f(m_0, m_1, ..., m_{n_1}) \\
\frac{\partial m_2}{\partial t} &= f(m_0, m_1, ..., m_{n_2}) \\
\frac{\partial m_3}{\partial t} &= f(m_0, m_1, ..., m_{n_3}) \\
&\vdots \\
\frac{\partial m_k}{\partial t} &= f(m_0, m_1, ..., m_{n_k}) \\
&\vdots \\
\frac{\partial m_{k_{max}}}{\partial t} &= f(m_{n_0}, m_{n_1}, ..., m_{n_{k_{max}}})
\end{aligned} \tag{3.11}$$

The right-hand side of each equation are the source terms. If the source term for the k^{th} moment transport equation only consists of moments of order up to k ($n_k \leq k$), the system is closed. In this case one solves the system starting with the first equation for the moment m_0. For all the subsequent equations the source terms are known from the solution of the previous equations and thus one can solve k_{max} differential equations in a straight forward manner. An example for solving such a system is provided in Sec. 3.3 where a simple homogeneous growth problem was considered.
However, if the source term for the k^{th} moment transport equation contains moments of order higher than k ($n_k > k$), the problem is unclosed. In this case the source term includes moments that cannot be computed from the previous equations, and thus, have to be calculated somehow differently.
An unclosed set of moment transport equations can be closed by several numerical meth-

ods, like the method of moments with interpolative closure (MOMIC), where the unknown moments are interpolated (or extrapolated) form the known set of moments [8, p. 293-295]. Since the numerical accuracy and the applicability of this methods is limited, a more reliable closure method is to use quadrature approximation. In this method the transport equations are closed by approximating the NDF based on the known moments up to level k. The Gaussian quadrature-based method of moments is presented in the following section.

3.2 Quadrature-Based Moment Methods

3.2.1 Gaussian quadrature

Gaussian quadrature is a numerical method to compute an integral property I of an NDF using the interpolation formula

$$I = \int_{\Omega_\xi} g(\xi) n d\xi = \int_{\Omega_\xi} \xi^k P_N(\xi) n d\xi \approx \sum_{\alpha=1}^{N} \xi_\alpha^k P_N(\xi_\alpha) w_\alpha \tag{3.12}$$

with the weights w_α and N the number of nodes ξ_α. $g(\xi)$ is a functional group, i.e., a so-called kernel or weight function [15], and P_N is a polynomial of degree N which is orthogonal to n. The condition for the application of this approximation is that $n(\xi)$ is continuous, and that $P_N(\xi)$ is a positive and integrable function with a finite number of roots.

It can be shown that in case the nodes ξ_α are the roots of the polynomial $P_N(\xi)$, Eq. (3.12) is **an exact relationship** for $k = 0$ through $k = 2N - 1$.
It is now important to realize, that the above relationship can be used to compute the moments of an NDF for a special choice of the polynomial P_N, namely $P_N(\xi) = 1$ [15], i.e.,

$$m_k = \int_{\Omega_\xi} \xi^k n d\xi \approx N_p \sum_{\alpha=1}^{N} \xi_\alpha^k w_\alpha \tag{3.13}$$

The sum of the weights yields one. Hence, the particle number concentration N_p is needed to take into account that n is not normalized. Based on the fact of exactness up to order $k = 2N - 1$, this means that moments up to order $k = 2N - 1$ can be computed **exactly** from N nodes. This degree of accuracy cannot be achieved by any other approximation, that is why the Gaussian quadrature is so attractive. (For example: in case one would use N nodes equally spaced in the integration interval, the degree of accuracy would be only $N - 1$.)

The nodes and weights can now be used to solve the closure problem. Specifically, the Gaussian quadrature approximation to the NDF is given by

$$n \approx N_p \sum_{\alpha=1}^{N} w_\alpha \delta(\xi - \xi_\alpha) \tag{3.14}$$

which means that the value of n at ξ_α is the infinite value of a delta function multiplied by the weight w_α, and zero otherwise. Thus, the NDF can be successfully reconstructed after computing ξ_α and w_α, and is approximated by a sum of Dirac-δ functions. For N

nodes one needs polynomials up to order $2N-1$, and thus the first $2N-1$ moments are required for the reconstruction based on the Gauss-quadrature approximation. This is the heart of the **quadrature method of moments** (QMOM).

As an example for the quadrature approximation we consider a normal distribution with $\sigma' = 1$ and $\mu' = 5$

$$n(\xi) = \frac{1}{\sqrt{2\pi}\sigma'} \cdot e^{-\frac{1}{2}\left(\frac{\xi-\mu'}{\sigma'}\right)^2} \tag{3.15}$$

The first eight moments of this equation can easily be calculated [8]:

$$\begin{aligned}
m_0 &= 1\\
m_1 &= \mu' = 5\\
m_2 &= \mu'^2 + \sigma'^2 = 26\\
m_3 &= \mu'^3 + 3\mu'\sigma'^2 = 140\\
m_4 &= \mu'^4 + 6\mu'^2\sigma'^2 + 3\sigma'^4 = 778\\
m_5 &= \mu'^5 + 10\mu'^3\sigma'^2 + 15\mu'\sigma'^4 = 4450\\
m_6 &= \mu'^6 + 15\mu'^4\sigma'^2 + 45\mu'^2\sigma'^4 + 15\sigma'^5 = 26140\\
m_7 &= \mu'^7 + 21\mu'^5\sigma'^2 + 105\mu'^3\sigma'^4 + 105\mu'\sigma'^6 = 157400
\end{aligned}$$

The weights and nodes, that are required for the quadrature approximation, can be calculated with the product-difference (PD) algorithm, that will be presented below. For our example we did so, using a PD-code in Matlab, and calculated the nodes and weights for $N = 3$ and $N = 4$. The calculated nodes and weights are listed in Tab. 3.1. It can be seen that the sum of the weights is always one. Note, since in this case $n(\xi)$ is normalized N_p is one.
Using Eq. (3.14) we can illustrate the NDF composed of N delta functions at the positions of the nodes, and weighted with the weights. In Fig. 3.1 this illustration is shown for the case of $N = 3$ and $N = 4$, as well as the exact NDF given by Eq. (3.15). (Be aware that this is just an illustration since the delta functions are infinite at the positions of the nodes!)
Note that the moments calculated form the approximated NDF give exactly the same values as for the exact case listed above.

Table 3.1: Nodes and weigths of the approximated normal distribution

$N = 3$		$N = 4$	
ξ_α	w_α	ξ_α	w_α
3,268	0,167	2,666	0,046
5,000	0,667	4,258	0,454
6,732	0,167	5,742	0,454
		7,334	0,046

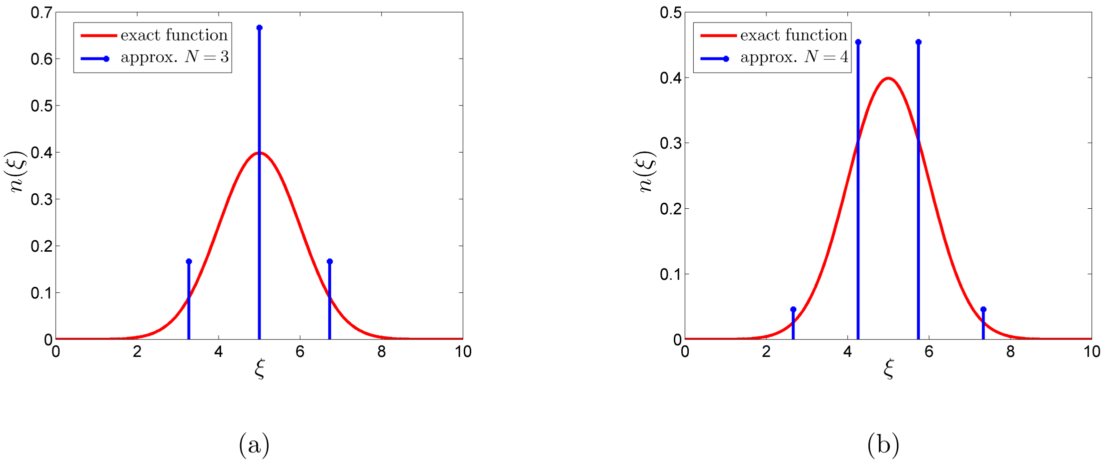


Figure 3.1: Illustration of a normal distribution function as NDF with $\mu' = 5$ and $\sigma' = 1$. The red line shows the exact function and the vertical blue lines represent the approximated NDF using Gaussian quadrature for (a) $N = 3$ and (b) $N = 4$ nodes.

To get the nodes and the weights required for the Gaussian quadrature approximation, a straight forward way to do so would be solving a nonlinear system of $2N - 1$ polynomial equations. Such an approach is not recommended [15], and next we describe a more sophisticated approach.

3.2.2 The Product-Difference (PD) Algorithm

Since the polynomials fulfill a recursion condition, this provides a smarter way to get the nodes and weights. The recursion

$$P_{\alpha-1}(\xi) = (\xi - a_\alpha)P_\alpha(\xi) - b_\alpha P_{\alpha-1}(\xi) \tag{3.16}$$

can be written in matrix form with a tridiagonal coefficient matrix, that can be symmetrized. This approach then allows one to get a system of equations in which the eigenvalues are the nodes and the weights can be calculated from the eigenvectors. This calculation can be done with the **product difference (PD) algorithm**. Since the polynomials are orthogonal with respect to n, the coefficients Eq. (3.16) are

$$a_\alpha = \frac{\int_{\Omega_\xi} n\xi P_\alpha(\xi)P_\alpha(\xi)d\xi}{\int_{\Omega_\xi} nP_\alpha(\xi)P_\alpha(\xi)d\xi} \qquad b_\alpha = \frac{\int_{\Omega_\xi} nP_\alpha(\xi)P_\alpha(\xi)d\xi}{\int_{\Omega_\xi} nP_{\alpha-1}(\xi)P_{\alpha-1}(\xi)d\xi} \tag{3.17}$$

By considering the expressions for the moments given by Eq. (3.13), it becomes clear that the coefficients a_α and b_α can be written in terms of the moments. In the PD algorithm this is exploited, and first a matrix $\boldsymbol{P}$ is constructed using the following algorithm.

- 1^{st} column: $P_{\alpha,1} = \delta_{\alpha 1}$ for $\alpha = 1, \ldots.., 2N + 1$
- 2^{nd} column: $P_{\alpha,2} = (-1)^{\alpha-1}m_{\alpha-1}$ for $\alpha = 1, \ldots.., 2N$
- remaining elements: $P_{\alpha,\beta} = P_{1,\beta-1}P_{\alpha+1,\beta-2} - P_{1,\beta-2}P_{\alpha+1,\beta-1}$ for $\beta = 3, \ldots.., 2N + 1; \alpha = 1, \ldots.., 2N + 2 - j$

The second step in the algorithm is to calculate the coefficients of the continued fraction $\{\zeta_\alpha\}$ using the relationship

$$\zeta_\alpha = \frac{P_{1,\alpha+1}}{P_{1,\alpha}P_{1,\alpha-1}} \qquad \text{for } \alpha = 2, \ldots 2N \tag{3.18}$$

The last step is the calculation of the coefficients via

$$a_\alpha = \zeta_{2\alpha} + \zeta_{2\alpha-1} \qquad \text{for } \alpha = 1, \ldots., N \tag{3.19}$$

$$b_\alpha = -\sqrt{\zeta_{2\alpha+1}\zeta_{2\alpha}} \qquad \text{for } \alpha = 1, \ldots., N - 1 \tag{3.20}$$

The PD algorithm becomes unstable for larger N (typically if $N > 10$). A more stable alternative is the **Wheeler algorithm**, that is described in [8, p. 53]. However, $N > 10$ is rarely used because of the larger number of moments that would need to be tracked. Consequently, we restrict ourselves to methods for which the PD algorithms is sufficiently stable.

3.2.3 The Correction Algorithm of McGraw

The moments that are used for calculating the weights and nodes of the Gaussian quadrature have to be realizable. This means that there has to exist an NDF that integrates to a set of moments used for the reconstruction of the NDF. In case the PD algorithm is fed with unrealizable moments, the calculation might become unstable.
Due to the fact that the moment transport equations are integrated numerically, there will always be some finite discretization errors which can lead to unrealizable moments. With the correction algorithm of McGraw the realizability is first checked, and unrealizable moments are corrected if necessary. The McGraw correction algorithm is based on generating a difference table with the difference vectors as columns, which is displayed for $N = 4$ in the following table:

$\boldsymbol{d_0}$	$\boldsymbol{d_1}$	$\boldsymbol{d_2}$	$\boldsymbol{d_3}$
$\ln(m_0)$	$\ln(m_1) - \ln(m_0)$	$\ln(m_2) - 2\ln(m_1) + \ln(m_0)$	$\ln(m_3) - 3\ln(m_2) - \ln(m_1) + \ln(m_0)$
$\ln(m_1)$	$\ln(m_2) - \ln(m_1)$	$\ln(m_3) - 2\ln(m_2) + \ln(m_1)$	0
$\ln(m_2)$	$\ln(m_3) - \ln(m_2)$	0	0
$\ln(m_3)$	0	0	0

The first step of the algorithm is the verification of the realizability of the moment set by checking the positivity of each element of the difference vector $\boldsymbol{d_2}$.
Then, in the case of a negative element of $\boldsymbol{d_2}$, a correction for the unrealizable moments is applied. As shown in [8, p. 55 - 60], an efficient way to perform the correction is to minimize the length of the third order difference vector $\boldsymbol{d_3}$.
The minimization procedure is based on changing the length of the vector $\boldsymbol{d_3}$ by changing a certain moment m_k by a factor c_k, and proving whether the length of $\boldsymbol{d_3}$ has reduced or not. Such a variation in the length of the third order difference vector $\boldsymbol{d_3}$ can be written as the difference between the unchanged vector $(\boldsymbol{d_3})_0$ and the changed vector $(\boldsymbol{d_3})_1$

$$(\boldsymbol{d_3})_1 - (\boldsymbol{d_3})_0 = \ln(c_k)\boldsymbol{b}_k \tag{3.21}$$

where $\boldsymbol{b}_k$ is the response vector that determines the number k of the modified element of $\boldsymbol{d_3}$. It is easy to see that the length of $(\boldsymbol{d_3})_1$ is minimal if it is orthogonal to $\boldsymbol{b}_k$. Therefore the length of $(\boldsymbol{d_3})_1$ is reduced by the variation (3.21) if the angle between $(\boldsymbol{d_3})_1$ and $\boldsymbol{b}_k$ is bigger than the angle between $(\boldsymbol{d_3})_0$ and $\boldsymbol{b}_k$. If this is the case, the corresponding moment is corrected by multiplying it with the correction factor.

$$(m_k)_1 = c_k \cdot (m_k)_0 \qquad \text{where} \qquad \ln(c_k) = \frac{\boldsymbol{b}_k \cdot (\boldsymbol{d}_3)_0}{||\boldsymbol{b}_k||^2} \tag{3.22}$$

3.2.4 Extended Quadrature Method of Moments (EQMOM)

With the QMOM the NDF is approximated by N weights and nodes (see Eq. 3.14), whereas for $N+1$ weights and nodes, another two moments would be required. This approach can be generalized by introducing a new way of representing the NDF:

$$n(\xi) \approx N_p \sum_{\alpha=1}^{N} w_\alpha \delta_\chi(\xi, \xi_\alpha) \tag{3.23}$$

where χ is fixed by one additional moment. For $\chi = 0$ the QMOM approximation is obtained. δ_χ can be found from the weight functions of a known family of orthogonal polynomials, like Laguerre [8, p. 83].
The advantage of EQMOM is that with one additional moment it is possible to reconstruct a smooth, non-negative NDF that exactly reproduces the first $2N+1$ moments.
Using a Gaussian distribution to define the kernel density function

$$\delta_\chi(x, y) = \frac{1}{\sqrt{2\pi}\chi} \exp\left(-\frac{(x-y)^2}{2\chi^2}\right) \tag{3.24}$$

the moments can be written as

$$m_k = m_k^* + \sum_{i=1}^{[k/2]} \frac{k!}{i!(k-2i)!} \left(\frac{\chi^2}{2}\right) m_{k-2i}^* \qquad \text{with} \qquad m_k^* = N_p \sum_{\alpha=1}^{N} w_\alpha \xi_\alpha^k \tag{3.25}$$

This system of $2N+1$ equations can be used to compute the N weights and nodes by iterating on χ^2. While this extended method has some advantages with respect to accuracy, this iteration requires additional computational cost. It is thought that EQMOM does not provide any significant advantage over QMOM in the context of this work, and hence we have not followed this approach in higher detail.

3.2.5 Direct Quadrature Method of Moments (DQMOM)

The QMOM and EQMOM requires the computation of a comparably large number of moments via the respective transport equations. Then, the nodes and weights are calculated, using, for example, the PD algorithms. The DQMOM follows another strategy, in which a moment-inversion algorithm is required only at the initialization of a computation. Thus, an algorithm is used to find the weights and nodes at time zero. Then, afterwards the time evolution for the nodes and weights is solved directly [8].
The DQMOM is illustrated for an univariate NDF considering a transport equation including advection (characterized by the particle velocity $\boldsymbol{v}$), diffusion (characterized by the particle diffusivity Γ), as well as a generic growth rate G. After computing the corresponding moment transport equations as shown above, and using the quadrature

approximation, one obtains the DQMOM transport equations for the nodes and weights

$$\begin{aligned} \frac{\partial w_\alpha}{\partial t} + \frac{\partial}{\partial x} u w_\alpha &= \Gamma \frac{\partial^2 w_\alpha}{\partial x^2} + a_\alpha \\ \frac{\partial}{\partial t} w_\alpha \xi_\alpha + \frac{\partial}{\partial x} u w_\alpha \xi_\alpha &= \Gamma \frac{\partial^2}{\partial x^2} w_\alpha \xi_\alpha + w_\alpha G(\xi_\alpha) + b_\alpha \end{aligned} \tag{3.26}$$

where a_α and b_α are the source terms for the weights and nodes, and $\alpha = 1, ..., N$. For $\Gamma >> 0$ the numerical solution of Eq. (3.26) will remain more accurate compared to a solution of Eq. (3.10). However, if Γ is very small or zero, the numerical solution becomes inaccurate.

When DQMOM is used to solve the moment equations, there might arise problems due to the fact that the nodes and weights (contrary to the moments) are not conserved in their transport equations. Thus, the weights and nodes could be discontinuous, even though the moment set is well defined. In fact, the DQMOM approach is likely to fail in the following cases [8, p. 338-340]:

(i) For purely hyperbolic moment-transport equations the DQMOM will fail if the nodes are discontinuous, which is usually the case when the velocity is one of the internal coordinates (i.e., in case of the GPBE). In this case QMOM must be applied.

(ii) For purely hyperbolic moment-transport equations without shocks in the nodes, the DQMOM will still fail if it is solved with a standard FVM because of inevitable numerical errors and numerical diffusion. To handle this problem one can use fully conservative DQMOM (DQMOM-FC).

(iii) For moment-transport equations with small diffusivities which are in the same magnitude as numerical diffusion (coming from the solution of the DQMOM equations using FVM), DQMOM should be avoided.

(iv) If the initial or boundary conditions are discontinuous (i.e. discontinuous moment set in the flow domain), DQMOM can fail if the transport equations are purely hyperbolic.

(v) If the moment sets are degenerated, DQMOM should be avoided due to the fixed value of N. This is for instance the case when a NDF is composed of a single weighted delta function and thus can be reconstructed by only two of the (well defined) moments.

In summary one can say that the advantage of the DQMOM is, that since the moment inversion algorithm is only used once, it is more accurate than QMOM. This is especially beneficial for a multivariate NDF, because in this case it is even more difficult to ensure the realizability of the moments. However, the DQMOM algorithm becomes inaccurate for very low diffusivities Γ. Moreover, since the weights and nodes in Eq. (3.26) are not conserved, conventional finite volume discretization schemes (see Sec. 3.4.2) might not be suitable for DQMOM. Because of the limitations and expected numerical difficulties, we have not further investigated the DQMOM.

3.3 Application of MOM and QMOM to Homogeneous Systems

For the homogeneous (where there is no spatial dependence) and univariate case, all derivatives with respect to x_i in the PBE become zero. Thus, one has to solve the following equation, that includes continuous and discontinuous changes in the internal coordinate:

$$\frac{\partial n(\xi,t)}{\partial t} + \frac{\partial}{\partial \xi}\dot{\xi} n = \mathcal{S}_1 \tag{3.27}$$

Depending on the physical models (see Sec. 3.5) that are used for the source terms the corresponding moment equations might be unclosed. The system of moment equations can be closed using different methods. Two of them are explained in the following sections.

3.3.1 Method of Moments

The principal idea of the method of moments was already discussed in Sec. 3.1.2. Here we just want to present an example.
We consider **condensation and growth of droplets** ($\dot{\xi} > 0$) using phase space drift

$$\frac{dm_k}{dt} = -\int_0^\infty \xi^k \left(\frac{\partial}{\partial \xi}\dot{\xi} n\right) d\xi \tag{3.28}$$

Applying integration by parts, the moment transport equation becomes

$$\frac{dm_k}{dt} = -\xi^k n\dot{\xi}(\xi)\Big|_0^\infty + k\int_0^\infty \xi^{k-1}\dot{\xi} n d\xi \tag{3.29}$$

For $k = 0$ we obtain

$$\frac{dm_0}{dt} = n(t,0)\dot{\xi}(0) = J(t,0) \tag{3.30}$$

where $J(t,0)$ is the nucleation rate under the assumption that nucleation occurs only at $\xi = 0$. Note, that in more realistic nucleation models nucleation is assumed to occur only above a critical droplet size, as it is shown in Sec. 3.5.4.
For $k > 0$ one can easily find a closed formulation of the problem for the case where $\dot{\xi}$ is independent of ξ:

$$\frac{dm_k}{dt} = k\dot{\xi} m_{k-1} \tag{3.31}$$

Otherwise, if $\dot{\xi}$ depends of ξ, the problem might be unclosed (e.g., for the power-law $\dot{\xi} = \dot{\xi}_0 \xi^\alpha$ the problem is unclosed for $\alpha > 1$) [8, p. 290-291].

For an unclosed set of moment transport equations the MOM fails because source terms of the moment transport equations depend on higher order moments which must be approximated. As already mentioned, a reliable closure method is to use quadrature approximation, that is discussed in the following section.

3.3.2 Quadrature-Based Moment Methods

As described in section 3.2, QMOM, DQMOM and EQMOM can be employed to overcome the closure problem.
DQMOM avoids the problems with the moment-inversion algorithm, since it is only applied once. The challenge in the case of DQMOM is to find the transport equations for the weights and nodes.
If the source term is highly localized in phase space, there is a risk of not having enough nodes in the localized regions. To overcome that problem, one could use EQMOM instead of increasing the number of nodes. [8, p. 300-305].

For **condensation and growth of droplets**, that was discussed in the former section, the moment transport equation Eq. (3.28) is written in the quadrature approximation (assuming $N_p = 1$) as follows:

$$\frac{dm_k}{dt} = k \sum_{\alpha=1}^{N} \xi_\alpha^{k-1} \dot{\xi}_\alpha w_\alpha \tag{3.32}$$

In the simple case for a constant growth rate ($\dot{\xi} \neq f(\xi)$), the solution is simply given by Eq. (3.31), with the initial condition

$$m_k(0) = \sum_{\alpha=1}^{N} w_\alpha(0) \xi_\alpha(0)^k \qquad k = 0, 1, ..., 2N-1 \tag{3.33}$$

To calculate the time evolution of the nodes and weights, one can apply the PD algorithm for each time step, as it can be found in [7] for the case of $N = 3$, where a simple growth problem was solved. A very similar problem we will treat in the following example.

Example: Homogeneous growth (reference problem)
As an example we want to consider a simple homogeneous growth problem that is given by Eq. (3.31). This example should act as a first reference problem for the implementation of the *qmomCloud* library. It is documented in [16], where it was solved with QMOM and DQMOM. Here it is solved using MOM and QMOM for the first 4 moments (i.e. $N = 2$).

The growth rate $\dot{\xi}$ in this example was considered as a power-law with 3 different exponents:

$$\dot{\xi}(\xi) = \xi^r \qquad with \qquad r = \begin{cases} 0 \\ 1 \\ -1 \end{cases} \tag{3.34}$$

The resulting moment equations thus are

$$\begin{aligned} \frac{dm_k}{dt} &= k \cdot m_{k-1} \qquad r = 0 \\ \frac{dm_k}{dt} &= k \cdot m_k \qquad r = 1 \\ \frac{dm_k}{dt} &= k \cdot m_{k-2} \qquad r = -1 \end{aligned} \tag{3.35}$$

Note that only for the case $r = 1$ the system of moment equations is closed, because for the other cases the low order equations include moments of negative order. (For the case $r = 0$ the equation for m_0 contains the moment m_{-1}, and for $r = -1$ the m_0-equation includes m_{-2} and the m_1-equation includes the moment m_{-1}.) Thus, basically only this closed case can be solved with MOM, where one just has to solve the ordinary differential equations in a straight forward manner).
However, one can find a closed form for the case $r = 0$ by setting $m_0 = const.$ This assumption is valid because for the simple growth case the number of droplets, and thus the corresponding moment m_0, does not change.
The only remaining unclosed case is that of $r = -1$. For this case we have to apply QMOM, while for the other two cases the system of equations can be solved with both, MOM and QMOM.

With QMOM the right-hand sides of Eq. (3.35) were approximated using Eq. (3.13).
In case we use the MOM, we just have to solve the ordinary differential equations Eq. (3.35) since the transport equations Eq. (3.31) were already given in the transformed form of the PBE. The differential equations were solved in Matlab using the explicit EULER method for the time integration.

For solving the time evolution of the moments we do of course need initial conditions. The initial moments were calculated from the initial NDF (see fig. 3.2), that, in this example, is given by

$$n(\xi, 0) = \begin{cases} 1 & \text{for } 0 \leq \xi \leq 1 \\ 0 & \text{for } \xi > 1 \end{cases} \tag{3.36}$$

The results were plotted as time evolution of the normalized Sauter mean diameter L_{32} (see Fig. 3.3) which is the ratio of the 4^{th} and the 3^{rd} moment.
Since the QMOM reproduces the exact moments for a closed set of moment equations, we obtained exactly the same results as for the MOM for the two closed cases. For the unclosed case $r = -1$, only the QMOM results were plotted, since there is no solution available with MOM.

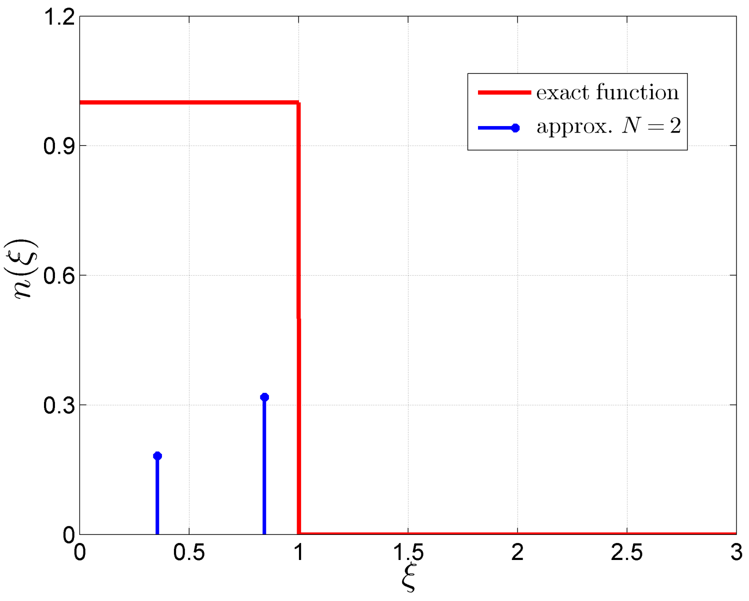


Figure 3.2: Initial NDF for the reference problem. The red line shows the exact distribution function, the blue vertical lines illustrate the NDF in the quadrature approximation for $N = 2$, according to Eq. (3.14).

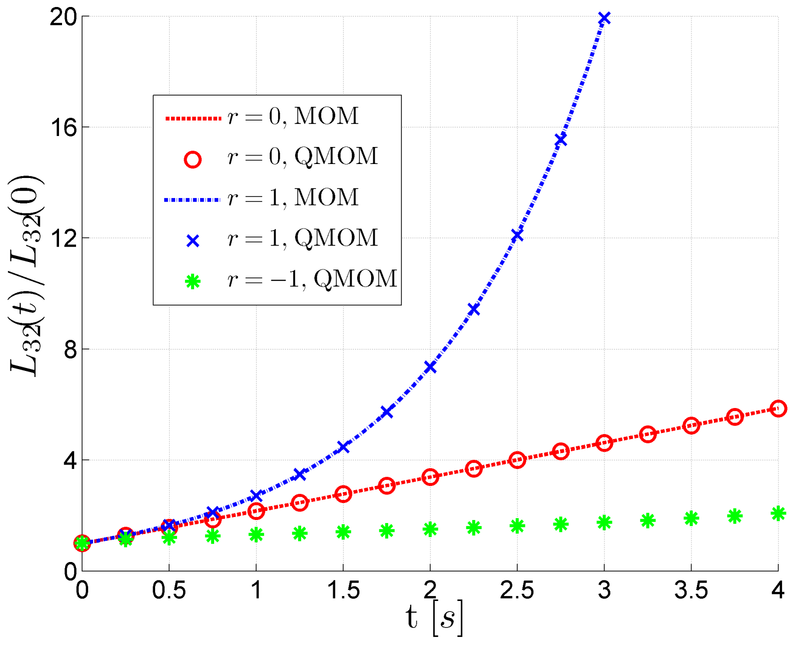


Figure 3.3: Time evolution of the normalized Sauter mean diameter for different exponents r of the power-law growth rate for the MOM case (dotted lines) and the QMOM case (symbols).

3.4 Closure Methods for Inhomogeneous Systems

3.4.1 Method of Moments

In inhomogeneous systems the particle (or droplet) cloud represented by the NDF (and thus its moments) moves in physical space due to advection and diffusion. For the simplest case of advection with a constant velocity $\boldsymbol{u}$ (and without changing the internal coordinates due to other processes), the change of the moments in physical space can be described by

$$\frac{\partial m_k}{\partial t} + u_i \frac{\partial m_k}{\partial x_i} = 0 \tag{3.37}$$

with the exact solution $m_k(\boldsymbol{x}, t) = m_k(\boldsymbol{x} - \boldsymbol{u}t, 0)$. Since the physical-space transport is calculated numerically, there are always numerical errors by solving such equations because of numerical diffusion.

In the case of QMOM, one uses a set of $2N$ moments to solve the moments transport equations, whereby only realizable moments can be used in the moment-inversion algorithm. The moments of the used moment set are correlated. Thus, the truncation of one moment due to numerical errors can lead to unrealizable moments. Therefore it is essential to guarantee the realizability of the moment set in every time step.

For an univariate PBE the realizability can be ensured by formulating the moment sets in terms of canonical moments or by using the McGraw correction algorithm to formulate realizable spatial transport schemes.

Moreover, to avoid excessive numerical diffusion, only high-order spatial-transport schemes should be employed [8, p. 330-332].

To compute the NDF and its moments one has to solve the corresponding transport equations (see Sec. 3.1.2).

If only the physical-space transport problem, accounting for advection and diffusion, is considered, the last two terms in Eq. (3.8) disappear (note, phase-space transport, e.g. growth, can be treated as shown in Sec. 3.3).

Depending whether the diffusion coefficient is a function of ξ or not, we get either a closed or an unclosed set of moment transport equations.

In the special case where $\Gamma \neq f(\xi)$, we obtain the closed form

$$\frac{\partial m_k}{\partial t} + u_i \frac{\partial m_k}{\partial x_i} = \frac{\partial}{\partial x_i}\left(\Gamma \frac{\partial m_k}{\partial x_i}\right) \tag{3.38}$$

In Sec. 3.5.2 a more realistic diffusion model is presented. In this model the diffusion coefficient does depend on ξ, resulting in an unclosed set of moment transport equations.

3.4.2 Numerical Schemes

To solve the spatial evolution of the moments, one uses finite-volume methods, whereas the changes of the internal coordinates are computed with the methods described in Sec. 3.3.
In finite-volume methods the physical space is divided into cells of size Δx leading to a uniform mesh. The spatial distribution of moments is represented as the volume-average of each cell j at the corresponding cell center $\boldsymbol{x}_j$.
This concept is applied to the moment transport equations Eq. (3.38). By using approximations (of different order) for the derivatives in the advection and diffusion terms, and appropriate time discretization (time step Δt), one can write the equations as a linear system of the form $\boldsymbol{A}\boldsymbol{m}^{n+1} = \boldsymbol{B}\boldsymbol{m}^n$. The **realizability condition** for the moments is, that all elements of the matrix $\boldsymbol{B}$ are nonnegative.
Standard finite-volume schemes (see [8], page 351 - 353) of order 4 or higher will almost always produce **unrealizable moments**. Therefore, realizable finite-volume schemes must be applied, which are based on reconstructing the NDF from their moments. The moment inversion algorithm is employed at each time step n for every grid cell i. In [8], p. 353, realizable finite-volume schemes are presented using gamma EQMOM for the moment inversion. The advantage of EQMOM is, that the approximated NDF reproduces the first $2N+1$ moments exactly .

However, one can also use QMOM to reconstruct the moments using Eq. (3.13). For an explicit time-stepping scheme, a first-order approximation for advection and a second-order approximation scheme for the diffusion (see [8], Section 8.3.2 and Appendix B.6), Eq. (3.38) becomes

$$m_{k,i}^{t+1} = \sum_{\alpha=1}^{N} w_{\alpha,i}^t \left(1 - 2c_{\alpha,i}^t\lambda_1 - \lambda_2\right)\left(\xi_{\alpha,i}^t\right)^k + \sum_{\alpha=1}^{N} w_{\alpha,i}^t \left(c_{\alpha,i}^t\lambda_1 - c^-\lambda_2\right)\left(\xi_{\alpha,i}^t\right)^k + \sum_{\alpha=1}^{N} w_{\alpha,i}^t \left(c_{\alpha,i}^t\lambda_1 - c^+\lambda_2\right)\left(\xi_{\alpha,i}^t\right)^k \tag{3.39}$$

with $\lambda_1 = \Gamma\Delta t/(\Delta x)^2$, $\lambda_2 = |u|\Delta t/\Delta x$, $c^- = (1 - u/|u|)/2$ and $c^+ = (1 + u/|u|)/2$.
and

$$c_{\alpha,i}^t = \begin{cases} 1 & \text{for constant diffusivity} \\ \xi_0/(\xi_{\alpha,i}^t + \xi_0) & \text{for Stokes-Einstein diffusivity.} \end{cases} \tag{3.40}$$

The constant ξ_0 in the Stokes-Einstein diffusivity is added to treat the limit of vanishing size so that the diffusivity coefficient remains finite.

The realizability condition in this case is

$$1 - 2c^t_{\alpha,i}\lambda_1 - \lambda_2 \quad \Rightarrow \quad \Delta t < \min_{\alpha,i}\left[\left(\frac{2c_{\alpha,i}\Gamma}{\Delta x^2} + \frac{|u|}{\Delta x}\right)^{-1}\right] \tag{3.41}$$

Thus, the realizability condition imposes a time-step limitation, similar to a Courant number limitation.

In the current work, OF is used to discretize the transport equations for the moments, and the numerical schemes for discretization of advection and diffusion terms can be selected at runtime (see [17]). The numerical schemes used are summarized in the corresponding results section (see Chapters 5 and 6).

3.5 Models for Physical Processes

In a granular system only particles in vacuum are considered that interact through collisions and low range forces with each other. In a two-phase system also the fluid (continuous phase) and its interactions with the disperse phase (particles) need to be taken into account. Such a polydisperse multiple phase model can be described by Eq. (B.7) which considers all possible changes of the NDF due to the interaction with the continuous phase, or particle-particle interactions.

In our case we focus on models to describe physical processes related to the condensation of vapor in a very dilute flow. Hence, particle-particle interactions are not important, and thus one can neglect many terms in Eq. (B.7). Furthermore we consider a population of particles (or droplets) that move more or less with the fluid, instead of tracking the trajectory of each particle. Thus we end up with the mesoscale description Eq. (3.8). With this equation all relevant physical processes that occur in a CPC can be calculated by using appropriate physical models, which will be presented in the following sections.

3.5.1 Phase Space Advection - Modeling Heat and Mass Transfer

Exchange of mass and heat between the disperse and the continuous phase leads to change in the internal coordinates. The driving force of mass transfer is a difference of the chemical potential of the vapor between the particle (or droplet) surface and the surrounding fluid (in our case a gas). For heat transfer, this would be a temperature difference between the two phases.

To derive an expression for the rate of change of the internal coordinate, a mass balance around the particle is considered [8]

$$\dot{m}_p = MW_v A_p j_{cond} \tag{3.42}$$

where MW_v is the molecular weight of the condensing or evaporating vapor (or more generally the exchanged molecules) that approach (or leave) the particle surface with area A_p with the flux j_{cond}.

In case of spherical droplets, their size and surface area can be described by only one internal coordinate, and we obtain [8]

$$\dot{\xi} = 2\frac{\dot{m}_p}{\rho_l \xi^2 \pi} \tag{3.43}$$

It remains to specify the rate of condensation $\dot{m}_p$, which needs to be closed by a phenomenological model for the condensation process.

Condensation on liquid droplets
Considering the mass transfer of vapor from a gas to a liquid droplet, we first assume that the vapor is treated to be in thermodynamic equilibrium on the droplet surface. This equilibrium can be described using, for example, the Clausius-Clapeyron equation (see [8], page 158) or more sophisticated expressions (see [10]). Mass transfer to the droplet surface (which is positive in case of condensation, and negative in case of evaporation) is modeled based on the model of Abramzon and Sirignano [18]

$$\dot{m}_p = \pi \rho D_v Sh \xi \ln(1 + B_m) \tag{3.44}$$

where ρ is the gas density, D_v is the diffusion coefficient of the vapor in the surrounding fluid, and $B_m = \frac{Y_{f1}-Y_{s1}}{1-Y_{s1}}$ is the Spalding mass transfer number. Y_{f1} is the mass fraction of evaporating molecules in the gas bulk (far away from the surface) and Y_{s1} is the mass fraction of vapor molecules near the surface (but in the gas phase). The latter can be computed from the assumed thermodynamic equilibrium at the surface. Sh is the Sherwood number, which is exactly two for the case of a spherical particle that does not move relative to the surrounding fluid. Note, a similar balance can be employed for the heat transfer to the particle surface [18]. Since we assume that the droplets are in thermal equilibrium with the surrounding fluid, this balance is not relevant for our work, though.

Using the above relations Eq. (3.43) and Eq. (3.44), we can write the moment transport equations for pure condensation as

$$\frac{\partial m_k}{\partial t} = \left[\frac{2}{\xi}\frac{\rho}{\rho_l} D_v\, Sh\, \ln(1 + B_m)\right] k\, m_{k-1} = \left[2\,\frac{\rho}{\rho_l} D_v\, Sh\, \ln(1 + B_m)\right] k\, m_{k-2} \tag{3.45}$$

Here, $G = \frac{2}{\xi}\frac{\rho}{\rho_l} D_v\, Sh\, \ln(1 + B_m)$ is the droplet growth rate.
Note that this is an unclosed set of equations that can be closed using QMOM.

3.5.2 Physical-space Diffusion - Modeling the Particles' Random Motion

Because of their small size, particles (or droplets) will exhibit a random motion triggered by the thermal energy of the surrounding fluid. This diffusive process can be described using the **Stokes-Einstein equation** [8, p. 187]

$$\Gamma = \frac{\Gamma_0}{d_p} = \frac{k_B T_f}{3\pi \mu_f d_p} \tag{3.46}$$

with the fluid temperature T_f, the fluid viscosity μ_f and the particle diffusion diameter $d_p = \xi$. The diffusion coefficient can be fed to Eq. (3.8).

A typical value of the Stokes-Einstein diffusion coefficient in a CPC would be about $2.44 \cdot 10^{-10}\,\frac{m^2}{s}$. Here a particle size of $100\,nm$, a temperature of $308,15\,K$ and a dynamic viscosity of $1.85 \cdot 10^{-5}\frac{kg}{m\,s}$ were used, according to a *3790_Gold CPC* from [10].

Note, that there is a counterpart of physical-space diffusion in the phase-space. This phase-space diffusion is related to fluctuations of the growth rate, typically observed in crystal growth processes. Since fluctuations of the growth rate have not been observed for the case of condensation on droplets, we have neglected this process.

Modeling only physical-space transport in this manner (where Eq. (3.46) is fed to Eq. (3.8)), the corresponding moment transport equations are

$$\frac{\partial m_k}{\partial t} + u_i \frac{\partial m_k}{\partial x_i} = \frac{k_B T_f}{3\pi\mu_f} \frac{\partial}{\partial x_i}\left(\frac{\partial m_{k-1}}{\partial x_i}\right) \tag{3.47}$$

Note that Eq. (3.47) is an unclosed form because the moment m_{-1} appears in the source term of the equation for m_0. Hence QMOM is required for this case as well.

3.5.3 Formation of a Disperse Phase - Homogeneous Nucleation

The formation of a disperse phase is generally related to molecular fluctuations that bring together a number of molecules. This process takes place in supersaturated continuous phases. In most cases (except when the new phase is locally and globally stable) an energy barrier has to be overcome, and thus, the new disperse phase (e.g., the droplet) is stable if it is larger than a critical size. The nucleation process is described by the source term on the right hand side of Eq. (3.8) and is categorized as a zero-order point process. Thus, the source term (fortunately) does not depend on the NDF. The nucleation process is accompanied by mass transfer from the continuous to the disperse phase, whereby the total mass and momentum of the former must be conserved. With the droplet size as internal coordinate and the assumption that droplets are formed at a critical size (see Sec. 3.5.4) the source term can be written as

$$\mathcal{S}_1 = J\delta(\xi - d_k) \tag{3.48}$$

where J is the nucleation rate of particles with the property d_k. It is assumed that the velocity of the particles and that of the fluid is equal, an assumption that we have already employed before. For the nucleation rate of monodisperse droplets of size d_k we can write

[8]

$$J = \frac{2D_v}{d_{mol}^5} \exp\left(-\frac{16\pi\sigma^3\tilde{\nu}^2}{3k_B^3T_f^3\left[\ln(S)\right]^2}\right) \tag{3.49}$$

Here D_v is the molecular diffusion coefficient of the vapor in the surrounding fluid, d_{mol} the molecular diameter, T_f the absolute fluid's temperature , S the supersaturation in the fluid and $\tilde{\nu}$ the molecular volume. σ is the surface tension between condensed vapor and fluid.
The critical formation size, also known as **Kelvin diameter**, can be estimated as

$$d_k = \frac{4\sigma\tilde{\nu}}{k_BT_f}\frac{1}{\ln(S)} \tag{3.50}$$

In CPCs, however, condensation occurs on nano meter sized particles, and hence the nucleation process is more complicated [12]. The following section will treat the heterogeneous nucleation process of a working fluid onto seed particles.

3.5.4 Nucleation of Droplets on a Particle Surface - Particle Activation

To initiate a growth process on primary particles (e.g., soot), the vapor has to form stable droplets that condense onto the particle surface. This is known as heterogeneous nucleation, contrary to homogeneous nucleation (discussed in the previous chapter) where the droplets are formed without condensation nuclei.
The easiest way to describe formation of new particles was shown in Sec. 3.3.1, where nucleation was assumed to take place at zero particle size. In reality, an energy barrier needs to be overcome before the formation of a droplet from vapor molecules can take place. Therefore only clusters with an adequately large size (i.e., a critical cluster size) are stable.
In [10] a mathematical description of the formation of vapor droplets on primary particles is provided. The calculations were done for butanol as working fluid, which is the case for many CPC's. Here, the governing equations of [10, p. 16-19] for the nucleation process of critical clusters from a gaseous working fluid and, in further consequence, for the activation process for growth on the seed particles will be summarized.

According to classical nucleation theory the energy barrier for the formation of a critical cluster (assuming a spherical cluster) is

$$\Delta G_{hom}^* = \frac{4}{3}\pi(r^*)^2\sigma \tag{3.51}$$

with the critical cluster radius

$$r^* = \frac{2\sigma MW_v}{\rho_l R_g T_f \ln(S)} \tag{3.52}$$

Here σ is surface tension between the vapor and the nucleated cluster that is assumed to be size independent. MW_v is the molecular weight of the working fluid and ρ_l is the density of the working fluid in liquid phase.
Note that the critical radius r^* is the half critical formation size d_k in Eq. (3.50) since $\tilde{\nu} = \frac{k_B MW_v}{R_g \rho_l} = N_A \frac{MW_v}{\rho_l}$.
In the heterogeneous nucleation theory it is assumed that a critical cluster is formed on the particle surface. Under the assumption of negligible line tension and size independent surface tension σ, the free energy of formation is

$$\Delta G^*_{het} = f_g \cdot \Delta G^*_{hom} \tag{3.53}$$

where f_g is a multiplicative factor (between 0 and 1) taking into account the contact angle θ between the critical cluster and the curved primary particle surface. Thus, all the chemical interactions between the primary particle and the forming vapor droplet (namely the particle affinity of the working fluid) are contained in the contact angle and hence in this factor. A contact angle of 180 indicates no affinity. In this case homogeneous nucleation occurs and $f_g = 1$.
A more detailed explanation of the multiplicative factor and its impact on the nucleation rate is given in Sec. A.

The nucleation rate can now be expressed in an Arrhenius form as [10]:

$$J_{het} = J^0_{het} \exp\left(-\frac{\Delta G^*_{het}}{k_B T}\right) \tag{3.54}$$

Note that if the kinetic prefactor $J^0_{het} = \frac{2D_v}{d^5_{mol}}$, the heterogeneous nucleation rate is the same as the nucleation rate out of [8, sec. 5] given by Eq. (3.49). In [10] the kinetic prefactor was calculated differently. However, this is of little consequence, since J^0_{het} has little impact on the nucleation process, that is anyhow very fast once the critical cluster radius (i.e., droplet size) is larger than the primary particle diameter. Nevertheless, for applications we will use the kinetic prefactor from [10], which is

$$J^0_{het} = d_p^2 \pi 10^{29} \tag{3.55}$$

Note, here the dimension of the factor $\pi 10^{29}$ is $m^{-5}s^{-1}$ in order to have consistent units. An important observation is that heterogeneous nucleation is much faster than homogeneous nucleation (see Appendix A for details). Consequently, it is very unlikely that under typical conditions in a CPC (i.e., supersaturations between 1 and 1.3) homogeneous nucleation takes place. Consequently, we simply limit the nucleation rate, such that the 0^{th} moment of the droplet population is always smaller than a specific primary particle concentration N_{prim}. Also, the diameter of the droplet in case of heterogeneous nucleation will be somewhat larger than that in homogeneous nucleation, simply because the droplet contains the primary particle. Hence, when computing the source terms in the moment equations, one needs to consider that droplets with an initial diameter

$$d_{d,init} = \sqrt[3]{d_p^3 + d_k^3} \tag{3.56}$$

form. In most practical cases, $d_{d,init}$ will be 1.26 times larger than the particle diameter, since d_k must be close to d_p to yield appreciable nucleation rates.

Applying this nucleation model, the moment transport equations for the pure droplet formation process is [19]

$$\frac{\partial m_k}{\partial t} = J_{het} \left(d_{d,init} \right)^k \tag{3.57}$$

Finally, note that in case the number concentration of droplets (or activated primary particles) is not tracked, a probability that a particle gets activated within a certain time must be specified [10]

$$P_{act}(t) = 1 - \exp\left(-J_{het} t \right) \tag{3.58}$$

The calculation of such an activation probability is not necessary when using MOM or QMOM, since we directly follow the evolution of the number concentration of the droplets. Note, that the fraction of activated droplets can be easily computed based on the droplet number concentration m_0 and the primary particle concentration N_{prim}.

3.6 Coupling Models

When nucleation and growth processes takes place in a CPC, mass transport from the continuous phase (= working fluid) to the disperse phase occurs. The consequent local depletion of the working fluid concentration must be taken into account in the source terms of the transport equations for the continuous phase.
Therefore, in this section the source terms for the species equation (= mass transport equation) and the heat transfer equation will be derived. In general, the condensation should lead to a local reduction of the saturation ratio due to the mentioned mass transfer. On the other hand, the local temperature should be increased due to release of condensation heat.

3.6.1 Mass Transport

The mass transport equation can be written as

$$\frac{\partial}{\partial t}\rho_v + \nabla \cdot (\rho_v \boldsymbol{u}_f) = -\dot{\mathcal{S}}_{coupl} \tag{3.59}$$

where ρ_v is the vapor density, $\boldsymbol{u}_f$ is the fluid velocity, and $\dot{\mathcal{S}}_{coupl}$ is a sink term describing mass transport from the continuous to the disperse phase.
Inside a CPC there is a certain amount of vaporized working fluid. Thus the vapor density can be written as a fraction y of the total gas density ρ:

$$\rho_v = y\,\rho \qquad \text{with} \qquad y = Y y_{sat} \tag{3.60}$$

Here Y is the dimensionless vapor mass fraction and y_{sat} is the saturated vapor mass fraction. Hence, the mass transport equation (also called species equation) is

$$\frac{\partial}{\partial t}(\rho Y) + \nabla \cdot (\rho Y \boldsymbol{u}_f) = -\frac{\dot{\mathcal{S}}_{coupl}}{y_{sat}} \tag{3.61}$$

Nucleation
According to the heterogeneous nucleation theory (see Sec. 3.5.4), the source term for the mass transport equation can be written as

$$\dot{\mathcal{S}}^{nuc}_{coupl} = \rho_l\, J_{het} V_p = \rho_l\, J_{het} \left(\frac{d_k^3 \pi}{6}\right) \tag{3.62}$$

Growth
For growth the source term can generally be written as

$$\dot{\mathcal{S}}^{growth}_{coupl} = \dot{m}_p\, N_p = \int_0^\infty \dot{m}_p(\xi)\, n\, d\xi \tag{3.63}$$

In the case of spherical droplets, where ξ is the droplet diameter and $m_p = \rho_l \frac{\pi \xi^3}{6}$, we can write

$$\dot{\mathcal{S}}_{coupl}^{growth} = \rho_l \frac{\pi}{6} \dot{m}_3 = \rho_l \frac{\pi}{6} \frac{d}{dt} \left(\int_0^\infty \xi^3(\xi)\, n\, d\xi \right) \tag{3.64}$$

Using $\frac{d}{dt} f(x) = \frac{df}{dx}\frac{dx}{dt} = \frac{df}{dx}\,\dot{x}$, the time derivative of the third moment becomes $m_3 = 3\,\dot{\xi}\,m_2$, and thus the source term is

$$\dot{\mathcal{S}}_{coupl}^{growth} = \rho_l \frac{\pi}{2}\,\dot{\xi}\,m_2 \tag{3.65}$$

The growth rate $\dot{\xi}$ is the mass that approaches the spherical droplet surface per time divided by the surface area. This behavior is expressed in Eq. (3.43).
With the growth model of Abramzon and Sirignano [18],the mass transfer to the disperse phase is given by Eq. (3.44). Thus, the growth rate is

$$\dot{\xi} = \left[2\,\frac{\rho}{\rho_l}\,D_v\,Sh \ln(1 + B_m) \right] \xi^{-1} \tag{3.66}$$

Finally, the source term for growth becomes

$$\dot{\mathcal{S}}_{coupl}^{growth} = \left[\rho\,\pi\,D_v\,Sh \ln(1 + B_m) \right]\, m_1 \tag{3.67}$$

Finally Taking into account both, the nucleation and the growth process, the whole mass transport equation is

$$\frac{\partial}{\partial t}(\rho Y) + \nabla \cdot (\rho Y \boldsymbol{u}_f) = -\frac{1}{y_{sat}} \left[\rho \pi D_v Sh \ln(1 + B_m)\, m_1 + \rho_l J_{het} \left(\frac{d_k^3 \pi}{6} \right) \right] \tag{3.68}$$

Note, as stated by Hinds [20], the size of the formed droplets has a minor effect to heat and mass transfer. Therefore, the impact on the transferred mass (and heat) due to nucleation is very low compared to that resulting from growth (see, Sec. 5.4).

With Eq. (3.68) the change in the vapor mass fraction can be calculated. Nevertheless, the more interesting quantity concerning nucleation and growth is the supersaturation. The supersaturation is defined as the ratio of the actual vapor concentration ρ_v and the saturation concentration of the vapor $\rho_{v,sat}$. When using the *cpcFoamCompressible* solver the quantities are referred to the values at the evaporator patch of the case (see Sec. 6.1). This means that at the evaporator the saturation ratio equals Y, and thus, we can write

$$S = \frac{\rho_v}{\rho_{v,sat}} = \frac{Y \rho_{v,sat}^{ref}}{\rho_{v,sat}} \tag{3.69}$$

Here, $\rho_{v,sat}$ is local concentration of the saturated vapor, that can be calculated assuming ideal gas behavior, via

$$\rho_{v,sat} = \frac{p_v(T)\, MW_v}{R_g\, T} \tag{3.70}$$

Here, p_v is the vapor pressure that depends on temperature. Finally we can write the supersaturation as

$$S = Y \frac{\rho_{v,sat}^{ref}\, R_g\, T}{p_v(T)\, MW_v} \tag{3.71}$$

3.6.2 Heat Transfer

Similar to Eq. (3.61) a transport equation for the energy of the continuous phase as a result of energy conservation can be formulated. In the entire heat transfer equation (see Appendix Eq. (B.6)) several terms, which are typically small, can be neglected, leading to

$$\frac{\partial}{\partial t}(\rho h) + \nabla \cdot (\rho \boldsymbol{u} h) = \nabla \cdot \left(\frac{\lambda}{c_p} \nabla h \right) + \dot{H}_{source} \tag{3.72}$$

Here, h is the specific gas enthalpy, λ is the heat conductivity and c_p is the specific heat. The source term for the heat transfer is related to that of the mass transfer via the vaporization enthalpy h_v due to nucleation and growth.
Thus, once the source terms for the species equation have been computed, the source for the energy equation is given by

$$\dot{H}_{source} = \dot{\mathcal{S}}_{coupl}\, h_v \tag{3.73}$$

The quantity of interest when changing the enthalpy of the system is the temperature which can be calculated via

$$T = \frac{h}{c_p(T)} \tag{3.74}$$

4 The *qmomCloud* Library

In this section the structure and a description of the main scripts of the *qmomCloud* library (in the following called *qmomCloud*) is presented. The *qmomCloud* is a part of the *cpcFoamCompressible* solver. Both solvers are written for the OpenFOAM® software package which will briefly be described in the following section.

4.1 OpenFOAM®

OpenFOAM(OF) is an open source CFD software package which provides a variety of standard solvers for fluid dynamics, heat transfer, multiphase flows, etc. The software is written in C++ and is designed for LINUX/UNIX operating systems [17].
OF has two types of applications: [21]

- solver: are applications to solve specific problems in continuum mechanics and are characterized by a respective differential equation (e.g. heat transfer equation)
- utilities: are applications to generate the geometry of the domain or to manipulate data

Beside the standard solvers, OF provides the possibility to implement own solvers, which is also the case for the *cpcFoamCompressible* solver (see Sec. 2).
A specific simulation case (i.e., a set of initial and boundary conditions, as well as physical and numerical parameters to be used as input for a solver) is defined in a file structure. Specifically, text files are used to control the solver, which can be dynamically updated (OF will monitor text files during the simulations). The most important files are:

- *blockMeshDict* utility: to define the test domain (geometry, boundaries, mesh, etc.)
- *controlDict* file: to select the solver to be used for the simulation, adjust time and time step settings, and define post-processing routines

An application is run by typing the application name in the LINUX shell. For instance, to generate the simulation domain, the `blockMesh` command must be used. To run the case, the name of the solver has to be used as command.

4.2 Architecture of *qmomCloud*

The purpose of the *qmomCloud* is to predict the time evolution of the moments of a particle size distribution by solving the corresponding transport equations. The set of equations to be solved is determined by the physical models employed to model the evolution of the particle size distribution (e.g., nucleation or growth of droplets). Specifically, a set of physical models can be selected at run time, and each physical model will add a source term to the moment transport equations. Additionally, the source terms for the coupling to the heat and (vapor) species transport equation are computed by the solver.

The software architecture of the *qmomCloud* library is sketched in Fig. 4.1. The philosophy of an object-oriented programming style (as dictated by C++ language) was used, and hence the code is split up into classes. These classes are used to generate objects, which are then able to handle data (e.g., read, store, exchange or write) and perform operations on the data. The core class of the *qmomCloud* library is the ***qmomCloud*** class. This top-level class manages all calculations, e.g., the function *evolve* will reconstruct the size distribution and solve the transport equation. Specifically, the ***qmomCloud*** class reads the user input (see Sec. 4.3), holds objects of all relevant sub-classes (e.g., for physical models), and exchanges data with the *cpcCompressible* solver. The *qmomCloud.H* and *qmomCloud.C* file contain the declaration and definition of the *qmomCloud* class, and can be found in the appendix (see Sec. E.1).

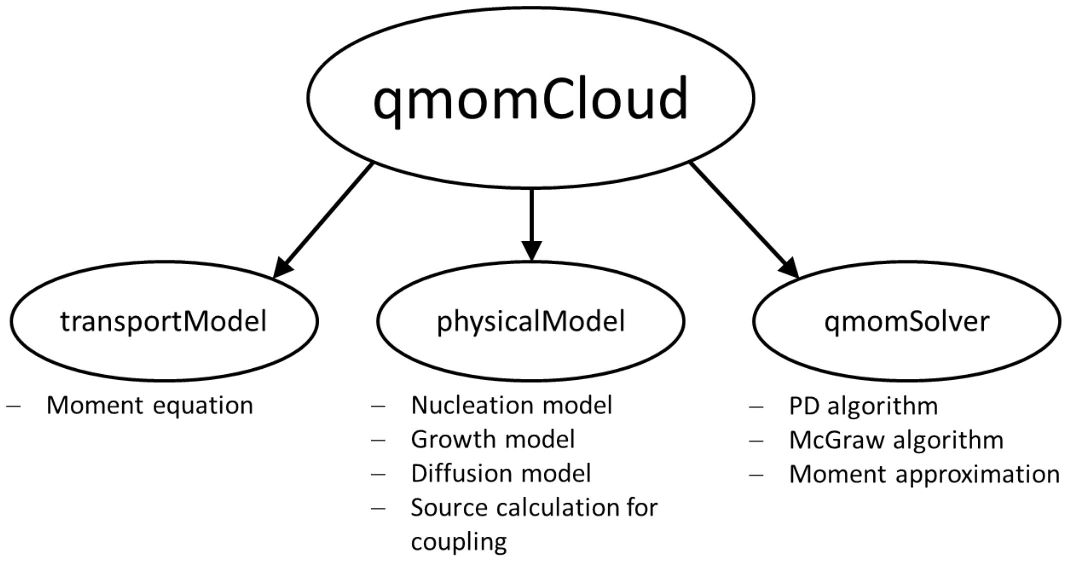


Figure 4.1: Architecture of the *qmomCloud*: The superordinated *qmomCloud* class manages the classes at the model and solver level.

The first subordinated class is the ***transportModel*** class. This class is responsible for solving transport equations relevant for predicting the particle size distribution. Currently, only one type of the *transportModel* class is implemented, namely the *momentTransport* class (see its definition in the *momentTransport.C*-file, Sec. E.1). In the syntax of OpenFOAM language, the moment transport equation is implemented in the following way:

```
//Assemble transport equation
fvScalarMatrix MEqn
(
    fvm::ddt(m_)
  + fvm::div(phi, m_,"div(phi,m)")     //phi is just velocity here
  ==
    fvm::laplacian(Dp_0_, m_,"laplacian(Dp,m)")
```

```
        + particleCloud_.momSource()
    );
```

The equation has the same form as Eq. (3.8), whereas the source term (i.e., the last term on the right hand side) is calculated by the *physicalModel* classes of *qmomCloud*, and handed over via the function *momSource*. The first term on the right hand side of the equation is only used if the Stokes-Einstein diffusion model is deactivated, i.e., a constant particle diffusivity is assumed (see Sec. 4.3).

The second subordinated class of the solver, called ***physicalModel***, manages the source terms for the moment equations (see Sec. 3.5), and for the coupling to the species and energy transport equation (3.6). Three physical models are implemented, and the definitions of the corresponding classes can be seen in the files *nucleation.C*, *simpleGrowth.C* and *diffusionPhysSpace.C*. All three files are attached in Sec. E.1. Note that source terms for the species and energy transport equations (i.e., the continuous phase) are only calculated for nucleation and growth, since diffusion in phyiscal space does not affect the continuous phase directly.

The third subordinated class is called ***qmomSolver***. This class is only executed in case the droplet size distribution needs to be reconstructed, i.e., if unclosed terms exist in the moment transport equation. For example, when using the growth model acc. to Sec. 3.5.1 or the Stokes-Einstein diffusion model of Sec. 3.5.2), unknown moments have to be computed from the reconstructed size distribution of the droplets, which is done in the *qmomSolver.C file* (see Sec. E.1). Currently, only one type of the **qmomSolver** class is implemented, namely the *productDifferenceQMOMSol* class (see the *productDifference-QMOMSol.C* file attached in Sec. E.1). This class contains routines to calculate nodes and weights in the context of the QMOM approximation using the PD algorithm (see 3.2.2). Furthermore, if the respective switch is activated, the moments are corrected via the McGraw correction algorithm (see Sec. 3.2.3).
Note, that the *qmomSolver* class itself contains routines to calculate the weights and nodes from the reconstructed particle size distribution acc. to Eq. (3.13). Thus, a specific implementation of a qmomSolver only has to provide the reconstructed size distribution.

4.3 How to Use the *qmomCloud* Library

In this section an overview is provided how a simulation case in OF is set up, and how it can be executed. The focus of the following discussion is on the use of the *qmomCloud* library. More details about the use of standard solvers, as well as more background with respect to other OF libraries and solvers can be found in [17]. Every simulation case in OF consists of a directory with a file structure as shown in Fig. 4.2, containing ASCII text files or binary files. Text files are used to store data, or control a simulation. The latter are called *dictionaries*, since OpenFOAM will look up settings in these files before and during a simulation run.

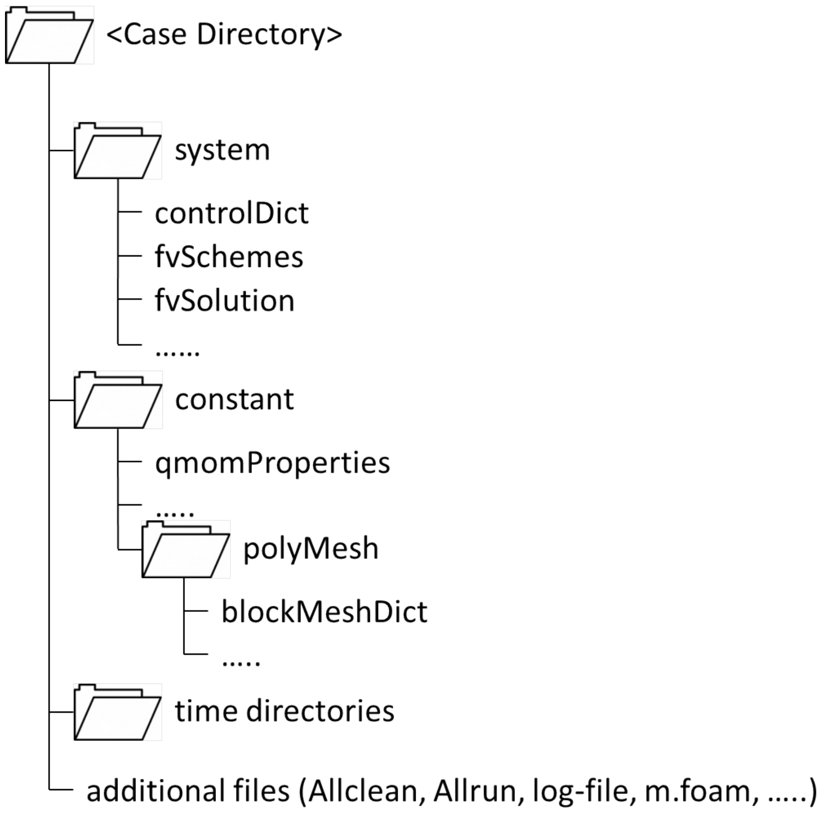


Figure 4.2: File structure of an OF case directory for use with the *qmomCloud* library

The ***'controlDict'*** file is the central dictionary to control the simulation run. For example, the desired solver for a simulation run can be specified. The specification of a solver is not necessary, however, it facilitates an automated execution of the case. Furthermore, the time settings (time step, start time, end time, etc.) and some other settings for data output can be specified in the *'controlDict'* file (the *'controlDict'* file is attached in Sec. E.2).

In order to allow a more flexible use of the *cpcFoamCompressible* solver, additional arguments that control the execution of certain routines within the solver will be read from the *'controlDict'* file. Specifically, four groups of routines of the *cpcFoamCompressible* solver can be activated or de-activated by specifying the corresponding key words in the first lines of the *'controlDict'* file. The following lines summarize the relevant part in the file:

```
application          cpcFoamCompressible; //compressible solver

minMaxOutputFreq     10; //frequency to report min/max statistics

//Main Switches for Solver Execution Control
```

```
solveFluidFlow;      //Activates the calculation of local velocity,
    pressure and density
solveEnergyEqn;      //Activates the calculation of the local enthalpy/
    temperature
solveSpeciesEqn;     //Activates the calculation of the local vapor
    concentration
solveParticleEqns;   //Activates the update of the particle cloud
    information
```

The first keyword, `'application'`, sets the solver to be used, i.e., the *cpcFoamCompressible* solver in our case. The first three of the sub-solvers can be activated by the keywords `'solveFluidFlow'`, `'solveEnergyEqn'` and `'solveSpeciesEqn'`. They activate the solution of the equations for the velocity-, temperature- and concentration fields of the continuous phase. The keyword `'solveParticleEqns'` activates all relevant calculations contained in the *qmomCloud* library, i.e., the calculation of the time evolution of the disperse phase by solving the moment transport equations.

In the ***'fvSchemes'*** dictionary the numerical schemes for individual terms in the transport equations are specified. For example, in this file the (numerical) approximations used to compute the divergence or gradient operators are specified. In the ***'fvSolution'*** file the error tolerances for solving linear systems of equations (required for all fields solved for) are specified. Also, other algorithmic details relating to the solution of the set of (discretized) governing equations are specified (for more details see [17]). Most important, when using the *qmomCloud* library, additional fields for the moments need to be defined in the *'fvSchemes'* and *'fvSolution'* files (the two files are attached in Sec. E.2). This is because transport equations for the moment set are solved by the library.

The ***'qmomProperties'*** file (see Sec. E.2) is the core dictionary that contains settings and parameters for the *qmomCloud* library. For example, the number of moments to be calculated, the physical models and their parameters, as well as the output of some intermediate results can be activated in this dictionary. Moreover, the product difference solver can be activated, which has to be done when the set of moment equations is unclosed. Note that whenever growth and physical-space diffusion is modeled, the system is unclosed, and thus the *PD* solver has to be activated.
The following settings must be specified before starting a simulation run that uses the *qmomCloud* library:

1. Number of Moments
   The *qmomCloud* provides a solution for the moment equations using up to three nodes and weights. This means that one can track up to six moments for approximating the droplet size distribution. In order to add or remove moments to be

tracked, simply a transportModel (e.g., *momentTransport*) needs to be added to the list of transport models (decativation can be simply done by commenting out in C/C++-style fashion or deletion):

```
transportModels          //choose the number of moments 2N you want
    to use, the number should be even in case the qmomSolver is used
(
    momentTransport //0
    momentTransport //1
    momentTransport //2
    momentTransport //3
//       momentTransport //4
//       momentTransport //5
);
```

2. Selection of Physical Models

   There are three physical processes that can be modeled: (i) nucleation, (ii) growth, and (iii) physical-space diffusion. Selecting a model is simply done by adding the name of the physical model to the list of physical models:

```
physicalModels
(
    simpleGrowth
//     nucleation
//     diffusionPhysSpace
);
```

3. Coupling Model

   The calculation of the source terms for the energy and the species equation can simply be done by including the keyword

```
couplingModel;
```

   Note, coupling does only make sense in case the species and energy equation is activated in the 'controlDict' file. Otherwise, the source terms for the continuous phase transport equations are computed, but the temperature and species (i.e., vapor) transport equations are not updated.

4. Activation of the *PD* solver

   To activate the Gaussian quadrature approximation for calculating the source terms of the moment equations, the product difference solver must be activated by

```
qmomSolver  productDifference;
```

   To deactivate the *PD* solver, `'qmomSolver'` is set to `'none'`.

5. Model Parameters

   To specify the parameters of the chosen models, their values has to be set in the corresponding sub-dictionary. The name of the sub-dictionary consists of the model name (e.g., `'nucleation'`) and the word `'Props'`. For example, settings for the nucleation model can be made using:

```
nucleationProps
{
//    verbose      true;
    SName        "S";
    theta        5; //in grad
    dParticle    1e-7;
    J0           1e5;
        NPrim            100;
        cellId2Report 4;
}
```

6. Output of Intermediate Results

   By setting the keyword `'verbose'` to `'true'` in a sub-dictionary, intermediate results will be displayed and hence can be routed to a log file. This is especially useful for testing and debugging models. For the output of the nucleation parameters, a specific cell can be chosen, which is done by setting the keyword `'cellId2Report'` to the value of the respective cell number.

7. Properties of the PD algorithm

   The PD algorithm might become unstable due to round-off errors and ill conditioned moments, especially in situations where moments become very small. Therefore, it is useful to set the calculation parameters for the moment inversion (i.e., PD) algorithm appropriately:

```
productDifferenceProps
{
//    verbose      true;
    iter_max              10;
    useCorrector          false;
    roundPrecision        1e-4;
    minConcentration      1e-6;
    computeEigenValues    true;
}
```

   In the presented example, with `'iter_max'` the iteration number in the moment update procedure is limited. With `'useCorrector'` the McGraw correction algorithm can be switched on, with `'roundPrecision'` the numerical accuracy of the moment inversion calculation procedure can be specified. `'minConcentration'` specifies a certain concentration under which the calculation of the nodes is suppressed by setting the nodes and weights to zero. By setting `'computEigenValues'` to `'false'` the eigenvalues, and thus, the nodes and weights, are guessed (and not computed) by assuming zero b-values in the PD algorithm. This setting should be avoided, unless the simulation suffers major instabilities caused by the PD algorithm.

The purpose of the ***'blockMeshDict'*** file is to generate geometry of the simulation domain, and perform the spatial discretization. The geometry can be generated by defining the spatial positions of vertices. Using these vertices, geometrical regions (so called 'blocks') of the domain can be defined (each block requires eight vertices to be defined). Afterwards the mesh is generated by discretizing the blocks into cells. Finally, the boundaries (i.e., exterior surfaces) are specified. For more detailed information see [17]. The *'blockMeshDict'* file of the pipe flow case (Sec. 6) is attached in Sec. E.2.

Before running the case, initial and boundary conditions need to be defined in the ***'0'***-directory. Here the initial values for all the fields (scalars and vectors in each cell of the domain) are defined. When using the *qmomCloud* library, the fields for the moments (m_0 to m_5) and for the three nodes and weights have to be defined in the *'0'*-directory as well.

Finally, a simulation run can be started by executing the **.\*Allrun***-script, which triggers mesh generation and the execution of the solver.

5 The 3x3-Test Case

To test the implementation of the *qmomCloud* library into the *cpcFoamCompressible* solver, a simple test case was set up. Specifically, the case consisted of a two-dimensional box, that was discretized into 3x3 cells in the x- and y-direction (see Fig. 5.1). In order to investigate the correct functions of the solver and the used models, several types of testing were performed. In these tests different conditions, like some specific initial NDF's, were chosen to evaluate the accuracy of the calculations done by the *qmomCloud* library.

In a first step, the correct implementation of the PD algorithm (including the McGraw correction algorithm) was tested. Afterwards the two physical models, *nucleation* and *simpleGrowth*, were verified. In order to be able to check the results of these two models, all other physical phenomena, like transport, were switched off. Therefore, in the *controlDict* file `'solveFluidFlow'`, `'solveEnergyEqn'` and `'solveSpeciesEqn'` were deactivated, and only `'solveParticleEqns'` was activated.
Finally the coupling models were tested in this simple test geometry in order to see how the saturation ratio and the temperature is affected by the nucleation and growth processes.

The original code (see [8, Appendix A1]) for the PD algorithm and the McGraw correction algorithm were available in Matlab® (ML). Therefore, the main functionalities of the QMOM method were tested in ML before it was implemented into the *qmomCloud* library and coupled with the *cpcFoamCompressible* solver.
Hence, the most test results provided in this chapter were compared to results calculated with ML, where no geometry, and thus, no spatial dependence was considered. Due to the fact that (spatial) transport phenomena were not considered, the results in every cell of our test geometry must yield the same results as the ML code.

There are three physical models implemented in the *qmomCloud* library, whereas one is physical-space diffusion. Since in this simple 3x3-test case transport was not modeled, and the implementation of the diffusion algorithm is straight forward and standard in OpenFOAM, the diffusion model was not tested.
As already mentioned in Sec. 3.4.2, high order numerical schemes for transport in physical space tend to produce unrealizable moments. However, since for the 3x3-test case there is no convective transport and no physical-space diffusion, satisfying the realizability condition, i.e., Eq. (3.39) is not problematic of this test case. Hence, the numerical schemes for the advection and diffusion terms are irrelevant.

5.1 Geometry of the 3x3-Test Case

The domain was chosen to be a simple 2D square that is divided into nine cells, three in each direction (see Fig. 5.1).

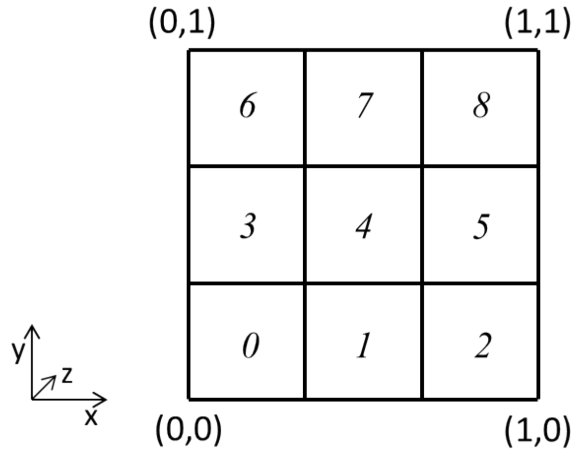


Figure 5.1: Illustration of the domain for the test geometry of the 3x3-test case. Each of the nine cells is labeled with its cell number.

To generate this geometry, the following settings were done in the *blockMeshDict* file:

- vertices: $(000), (100), (110), (010), (0\,0\,0.1), (1\,0\,0.1), (1\,1\,0.1), (0\,1\,0.1)$
- blocks: $hex(01234567)(331)simpleGrading(111)$
- patches: *evaporator* (left side), *condenser* (right side), $fixedWalls$ (top and bottom), $frontAndBack$

Note that the patches `'evaporator'`, and `'condenser'` are required by the *cpcFoam-Compressible* solver, and MUST be specified. This is due to the fact that the solver *cpcFoamCompressible* searches for these patches to compute reference quantities needed during the solution procedure.
The patch `'frontAndBack'` was set to `'empty'` to enable a 2D treatment of the case.
The boundary fields of all the initial field files in the *'0'*-directory of the case have to be adapted according to the patches in the *blockMeshDict* file.

5.2 Reconstruction of the Size Distribution using *qmomCloud*

The heart of the *qmomCloud* is the reconstruction of the size distribution, i.e., calculation of the nodes and weights using the PD algorithm. The PD algorithm uses $2N$ moments to first construct the matrix $\mathbf{P}$. Then, using the elements of the matrix $\mathbf{P}$, the N weights and nodes are computed by calculating the eigenvalues of a Jacobi matrix (see 3.2.2).
The calculation of the weights and nodes in the *qmomCloud* is done in the *'qmomCloud/-subModels/qmomSolver/productDifferenceQMOMSol.C'* file, which will be denoted as *PD* solver in the following discussion.
The number of moments that should be tracked determines the number of nodes and weights that are used for the QMOM approximation (see 3.2). For N nodes and weights, the required number of moments is $2N$. Thus, only an even number of moments can be used.
The nodes and weights are calculated by solving an eigenvalue problem. Since the standard eigenvalue solver in OF is limited to six eigenvalues, at most three nodes and weights can be used. Therefore the possible number of moments is limited to maximum six, corresponding to up to three nodes. However, as stated in [22, p.51], with $N = 3$ the achievable accuracy is still sufficiently high. The restrictions were implemented by producing an error that aborts the computation when using a number of moments greater than six.

The correct implementation of the *PD* solver was tested based on Exercise 3.1 in [8], with initial moments (that have to be set in the *zero*-directory of the case):

m_0	m_1	m_2	m_3	m_4	m_5
1	5	26	140	778	4450

In this example all the intermediate results in OF were compared to the reference results computed in ML. This was done for $N = 2$ and $N = 3$. In Tab. 5.1 it can be seen, that perfect agreement between the ML and the OF results was found, indicating the correct implementation of the PD algorithm.

As mentioned in Sec. 3.2.3, the set of moments might become unrealizable due to numerical errors. To avoid unrealizable moments the McGraw correction algorithm was implemented in the *PD* solver. In case the correction algorithm should be used, it has to be activated in the *constant/qmomProperties* file. This is done by setting the variable `'useCorrector'` to `'true'` in the `'productDifferenceProps'` sub-dictionary.

Table 5.1: Comparison of the results of Exercise 3.1 from [8]. The intermediate results of the *PD* solver and the obtained moments in OF were checked against the ML code for $N = 3$.

	Matlab	OpenFOAM
matrix **P**	$\begin{pmatrix} 1 & 1 & 5 & 1 & 24 & 10 & 1100 \\ 0 & -5 & -26 & -10 & -250 & -150 & 0 \\ 0 & 26 & 140 & 78 & 2022 & 0 & 0 \\ 0 & -140 & -778 & -560 & 0 & 0 & 0 \\ 0 & 778 & 4450 & 0 & 0 & 0 & 0 \\ 0 & -4450 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 11 & 1 & 5 & 1 & 24 & 10 & 1100 \\ 0 & -5 & -26 & -10 & -250 & -150 & 0 \\ 0 & 26 & 140 & 78 & 2022 & 0 & 0 \\ 0 & -140 & -778 & -560 & 0 & 0 & 0 \\ 0 & 778 & 4450 & 0 & 0 & 0 & 0 \\ 0 & -4450 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$
Jacobi matrix **J**	$\begin{pmatrix} 5 & -1 & 0 \\ -1 & 5 & -1.414214 \\ 0 & -1.414214 & 5 \end{pmatrix}$	$\begin{pmatrix} 5 & -1 & 0 \\ -1 & 5 & -1.414214 \\ 0 & -1.414214 & 5 \end{pmatrix}$
nodes $(\xi_1\ \xi_2\ \xi_3)$	$\begin{pmatrix} 3.2679 & 5.0000 & 6.7321 \end{pmatrix}$	$\begin{pmatrix} 3.267949 & 5 & 6.732051 \end{pmatrix}$
weights $(w_1\ w_2\ w_3)$	$\begin{pmatrix} 0.1667 & 0.6667 & 0.1667 \end{pmatrix}$	$\begin{pmatrix} 0.1666667 & 0.6666667 & 0.1666667 \end{pmatrix}$
moments $\begin{pmatrix} m_0 & m_1 & m_2 \\ m_3 & m_4 & m_5 \end{pmatrix}$	$\begin{pmatrix} 1 & 5 & 26 \\ 140 & 778 & 4450 \end{pmatrix}$	$\begin{pmatrix} 1 & 5 & 26 \\ 140 & 778 & 4450 \end{pmatrix}$

Also, the correction algorithm was tested and compared to the results of the ML routine that is based on the McGraw correction algorithm in [8]. This ML code yields the correct results for Exercise 3.4 in [8]. Thus, the implementation of the correction algorithm was tested according to this example, where the used moment set was

m_0	m_1	m_2	m_3	m_4	m_5
1	5	26	101	778	4450

Here m_3 is changed compared to the previous moment set.
Also for this example all the intermediate results of the correction algorithm were tested to ensure the correct implementation. The results for $N = 3$ are listened in Tab. 5.2.
We found no deviations from the ML results, indicating the correct implementation of the McGraw correction algorithm.

Note, for realistic applications (i.e. to model the formation of a disperse phase in a CPC), there is no initial NDF. This means that the initial moments are all zero, and the disperse phase is formed once suitable conditions for nucleation exist. The size distribution of the formed disperse phase is represented by the moment set. Thus, it is important that our solver produces a realizable moment set when nucleation takes place. For this reason, the McGraw correction algorithm was implemented to guarantee realizability requirement.

Table 5.2: Comparison of the intermediate results of Exercise 3.4 from [8] for $N = 3$ to test the McGraw correction algorithm.
The vector $\mathbf{d_i}$ is the i^{th}-order difference vector, k^* is the index of the moment that has to be changed, c_k is the correction factor of the McGraw correction algorithm, and m_k are the corrected moments. Note that in ML the index of the moment m_0 is 1. Thus the index for the same moment is always one higher than for the OF case.

	Matlab	OpenFOAM
$(\boldsymbol{d_0}\ \boldsymbol{d_1}\ \boldsymbol{d_2}\ \boldsymbol{d_3})$	$\begin{pmatrix} 0 & 1.6094 & 0.0392 & -0.0042 \\ 1.6094 & 1.6487 & 0.0350 & -0.0037 \\ 3.2581 & 1.6836 & 0.0312 & -0.0023 \\ 4.9418 & 1.7149 & 0.0289 & 0 \\ 6.6567 & 1.7440 & 0 & 0 \\ 8.4010 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 1.6094 & 0.0392 & -0.0042 \\ 1.6094 & 1.6487 & 0.0350 & -0.0037 \\ 3.2581 & 1.6836 & 0.0312 & -0.0023 \\ 4.9418 & 1.7149 & 0.0289 & 0 \\ 6.6567 & 1.7440 & 0 & 0 \\ 8.4010 & 0 & 0 & 0 \end{pmatrix}$
k^*	4	3
$\ln(c_k)$	0.3308554	0.3308554
number of iterations	1	1
$\begin{pmatrix} m_0 & m_1 & m_2 \\ m_3 & m_4 & m_5 \end{pmatrix}$	$\begin{pmatrix} 1 & 5 & 26 \\ 140.0174 & 778 & 4450 \end{pmatrix}$	$\begin{pmatrix} 1 & 5 & 26 \\ 140.0174 & 778 & 4450 \end{pmatrix}$

5.3 Physical Models: Nucleation and Growth

The tests for the correct implementation and functionality of the physical models, *nucelation* and *simpleGrowth*, was based on calculating the time evolution of the moments. The two physical models are presented in Sec. 3.5. Additionally a simpler growth model was implemented, that was already applied in the reference problem in Sec. 3.3.2. This simple model was considered because of its low number of parameters, and the ability of comparing the results to that in [16]. For realistic modeling of a CPC, however, this simple growth model is not of interest.

To monitor the moment set, as well as the required fields (e.g. Y, S), the corresponding fields were probed in the cell located in the middle of the domain (cell number 4). This means that the field values at a specific point (in the middle of the domain) were recorded for every time step in order to get the time evolution of the quantities. Beside the probe settings, other preferences regarding to simulation time and the choice of solvers were done in the *controlDict* file.

Note that the time settings can vary for specific test cases to ensure realistic conditions. In order to avoid that the calculations become inaccurate when changing the time settings, the variable `'adjustTimeStep'` must be set to `'no'` in the *controlDict* file.
We have combined the implemented models in a way that four different test cases were performed.

5.3.1 Case 1 - Simple Growth with Power-law Growth Rate (Reference Problem)

In this first test case the moment equations have the following form:

$$\frac{\partial m_k}{\partial t} = k\, G_0\, \xi^r\, m_{k-1} = k\, G_0\, m_{k+r-1} \tag{5.1}$$

Depending on the exponent r we get either a closed or an unclosed set of equations. For $r = 1$ the system is closed and we can calculate the time evolution without QMOM. For $r = 0$ the system is unclosed for the moment m_0, however, for the simple growth case we can set $m_0 = const.$ since the number of droplets does not change. (Moreover $m_0 = const.$ anyway since the source term vanishes because $k = 0$.)
Thus, for these two cases no closure method (QMOM in our case) is required and the time evolution of the moments can be calculated by any ordinary differential equation solver. In OF this is automatically done when using the *cpcFoamCompressible* solver.
To verify the implementation of the *qmomCloud* library, we also calculated the simple growth example in ML using the explicit Euler-method for the time integration.

According to the simple homogeneous growth problem in [16], the initial moment set in is

m_0	m_1	m_2	m_3
1	0.5	0.33334	0.25

To select this simple growth model with the power law growth rate G_0 and the exponent r, `'useLocalG'` must be commented out in the *'qmomProperties'* file. Moreover, the values for G_0 and r are set in this file. Specifically, we choose $G_0 = 1$, $r = 1$, $r = 0$ and $r = -1$.

For the two closed cases ($r = 1$ and $r = 0$) the solver was first tested without using the PD algorithm. For this test excellent agreement with the reference data (i.e., the ML results) was found.
Afterwards all three cases, $r = 1$, $r = 0$ and $r = -1$, were calculated and the probed moments (in cell number 4) were compared to the ones calculated with ML. Tab. 5.3 shows the values of the calculated moments for all three power-law cases after the first time step. The relative error between the OF and the ML results is at most 0.12%, which is satisfactory small. After four seconds, the relative error did not grow significantly, as it can be seen in Tab. 5.4.

In Fig. 5.2 the time evolution of the Sauter mean diameter was plotted. The OF results show excellent agreement with the ML results for both, the closed and the unclosed cases.

Table 5.3: Comparison of the moments after the first time step (at $t = 2 \cdot 10^{-4}\,s$) calculated using QMOM in ML and in OF, for the reference problem (power-law cases r=1, r=0 and r=-1). The relative error between the ML and the OF results was calculated by dividing the differences of the two values by the ML value.

	OpenFOAM			Matlab			rel. error		
	r=1	r=0	r=-1	r=1	r=0	r=-1	r=1	r=0	r=-1
m_0	1,0000000	1,0000000	1,0000000	1,0000000	1,0000000	1,0000000	0,00%	0,00%	0,00%
m_1	0,5001000	0,5002000	0,5006001	0,5002000	0,5004000	0,5011978	0,02%	0,04%	0,12%
m_2	0,3334700	0,3335400	0,3337400	0,3336001	0,3337334	0,3341333	0,04%	0,06%	0,12%
m_3	0,2501500	0,2502000	0,2503000	0,2503001	0,2504001	0,2506004	0,06%	0,08%	0,12%

Table 5.4: Comparison of the moments after the first time step (at $t = 4\,s$) calculated with QMOM in ML and in OF, for the reference problem (power-law cases r=1, r=0 and r=-1). The relative error between the ML and the OF results was calculated by dividing the differences of the two values by the ML value.

	OpenFOAM			Matlab			rel. error		
	r=1	r=0	r=-1	r=1	r=0	r=-1	r=1	r=0	r=-1
m_0	0,9991118	0,9991118	0,9991118	1,0000000	1,0000000	1,0000000	0,09%	0,09%	0,09%
m_1	27,2639200	4,4960030	2,8837720	27,2936167	4,5002000	2,8864040	0,11%	0,09%	0,09%
m_2	99,1203200	20,3144800	8,3259380	99,2461338	20,3343333	8,3337333	0,13%	0,10%	0,09%
m_3	40506,5300000	92,1573500	24,0453300	40566,8657680	92,2513998	24,0684078	0,15%	0,10%	0,10%

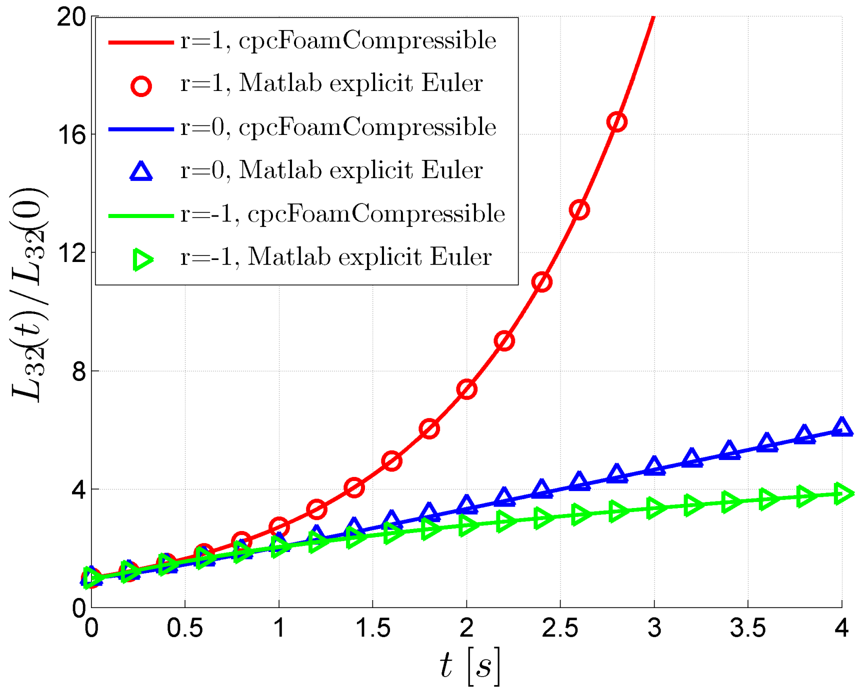


Figure 5.2: Time evolution of the normalized Sauter mean diameter for all power-law cases, $r = 1$, $r = 0$ and $r = -1$, of the reference problem. The calculation was performed using the explicit Euler method in ML (solid lines) and with the *cpcFoamCompressible* solver in OF (symbols).

These tests showed the correct implementation of the power-law growth model on the one hand, and, so far, the correct functionality of the *PD* solver as a closure method on the other hand. Hence, we continued to test the more sophisticated growth model according to Abramzon and Sirignano [18], which will be discussed in the following section.

5.3.2 Case 2 - Growth according to Abramzon and Sirignano

For this more realistic growth model [18] the moment equations are

$$\frac{\partial m_k}{\partial t} = 2\,\frac{\rho}{\rho_l} D_v\, Sh\, \ln(1+B_m)\, k\, m_{k-2} \tag{5.2}$$

To test the growth model, we assumed that at time zero there is one particle per cubic meter of size d_p. Since this represents a monodisperse distribution, the initial NDF can be written as

$$n(\xi) = \delta(\xi - d_p) \tag{5.3}$$

and thus, the moments are

$$m_k = \int_0^\infty \xi^k \delta(\xi - d_p)\, d\xi = d_p^k \tag{5.4}$$

In order to deal with realistic conditions according to that in a CPC, the initial droplet size d_p was set to $50\,nm$ for the following tests of the growth model. Specifically, the initial moments were

m_0	m_1	m_2	m_3	m_4	m_5
1	5E-08	2.5E-15	1.25E-22	6.25E-30	3.125E-37

As already mentioned, this growth case requires the reconstruction of the source term using QMOM. To do so, the variable `'useLocalG'` must be activated in the *qmomProperties* file. As before, the presented OF results originate from the probe in cell number 4. Additionally, the following settings were used:

(i) main settings in the *'constant/qmomProperties'* file:
```
useLocalG;
```
(ii) time settings in the *'system/controlDict'* file:
```
deltaT 1e-7;
writeControl adjustableRunTime;
writeInterval 1e-7;
```
(iii) gas density in the *'constant/thermophysicalProperties'* file:
```
equationOfState { rho 1.1; }
```
(iv) liquid density by choosing butanol as working fluid in the *'constant/transportProperties'* file:
```
binaryDiffusivityModel constant;
diffusingSpecies C4H10On;
diffusingIn air;
```
(v) initial conditions in the *'0'*-directory:

-) mass fraction: `Y = 1.1;`

-) temperature: `T = 319.15;`

With these settings butanol was chosen as working fluid. The *cpcFoamCompressible* solver calculates the required species quantities according to Sec. C of the Appendix, resulting in an initial growth rate of $G = \frac{2}{d_p} \frac{\rho_g}{\rho_l} Sh\, D_v \ln(1 + B_m) = 1.08478 \cdot 10^{-2} \frac{m}{s}$.

For the moment set of a monodisperse NDF, the original PD algorithm from [8] fails due to a division through zero when calculating the coefficients of the continued fraction (see Sec. 3.2.2). This behavior of the algorithm was contrary to our expectations and the incorrect statement in [8, p. 53] that a failure only occurs when dealing with distributions that have a zero mean value (i.e. m_1=0).
To avoid infinite values, and thus, meaningless nodes and weights, the PD algorithm from [8] was modified. Moreover, the calculation of complex nodes and weights was undermined in this modified version of the PD algorithm.

In order to proof the reliability of the *PD* solver, the modifications were first done in the ML-code. With the explicit Euler algorithm the the moment update is

$$m_k^{t+1} = m_k^t + k \cdot \Delta t \cdot G \cdot d_p \cdot m_{k-2}^t \tag{5.5}$$

Since the *PD* solver had problems with handling a monodisperse distribution, we first used the following moment update procedure:

$$m_k^{t+1} = m_k^t + k \cdot \Delta t \cdot G \cdot d_p \cdot d_p^{k-2} \tag{5.6}$$

Note, this procedure is only valid for the first update, because the moments have changed after the first time step. However, under the assumption of a monodisperse distribution the procedure can also be applied for further time steps by setting the droplet diameter d_p to the value of the (arithmetic) mean particle size m_1/m_0. This allows to compare the results from the modified PD algorithm with a solution were no closure is required.

However, beside the mentioned problems, further changes in the *PD* solver had to be done in order to ensure stability when dealing with a monodisperse distribution. For example, deviations from the ideal shape of the matrix $\boldsymbol{P}$ for the monodisperse case due to numerical round-off errors were found.

The analytical expression for the matrix $\boldsymbol{P}$ in case of a monodisperse distribution is:

$$P = \begin{pmatrix} 1 & 1 & d_p^1 & 0 & 0 & 0 & 0 \\ 0 & -d_p^1 & -d_p^2 & 0 & 0 & 0 & 0 \\ 0 & d_p^2 & d_p^3 & 0 & 0 & 0 & 0 \\ 0 & -d_p^3 & -d_p^4 & 0 & 0 & 0 & 0 \\ 0 & d_p^4 & d_p^5 & 0 & 0 & 0 & 0 \\ 0 & -d_p^5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Tab. 5.5 shows the matrices calculated in ML and OF for a monodisperse particle of size $d_p = 50\,nm$. The round-off errors lead to deviations (that are different for ML and OF) which cause instabilities in the further calculation procedure of the PD algorithm. Specifically, non-zero entries in colums four and up are notices, which result from the difference of two numbers of similar magnitude that are not equal.

Table 5.5: Comparison of matrices **P** calculated in ML and in OF after the first time step.

	1,0000E+00	1,0000E+00	5,0000E-08	0	1,18E-45	-5,6291E-84	-2,76E-83
	0	-5,00E-08	-2,50E-15	-2,35E-38	0	-5,4216E-91	0
	0	2,50E-15	1,25E-22	0	-2,09E-60	0	0
Matlab	0	-1,25E-22	-6,25E-30	4,18E-53	0	0	0
	0	6,25E-30	3,13E-37	0	0	0	0
	0	-3,13E-37	0	0	0	0	0
	0	0	0	0	0	0	0
	1,0000E+00	1,0000E+00	5,0000E-08	3,9443E-31	-9,8608E-46	-5,6291E-84	-6,1494E-136
	0	-5,0000E-08	-2,5000E-15	0,0000E+00	1,4271E-53	-5,4216E-91	0
	0	2,5000E-15	1,2500E-22	7,0065E-46	-3,7709E-61	0	0
OpenFOAM	0	-1,2500E-22	-6,2500E-30	-4,1762E-53	0	0	0
	0	6,2500E-30	3,1250E-37	0	0	0	0
	0	-3,1250E-37	0	0	0	0	0
	0	0	0	0	0	0	0

Consequently, the following modifications of the *qmomCloud* library were made:

- Since the deviations of the ideal results originate from rounding errors, the round precision in the calculation procedure was made to be adjustable. The adjustment is done in the *'qmomProperties'* file by setting the variable `'roundPrecision'`.
- The new *PD* solver checks whether the sum of the weights equals unity. If the sum is not exactly unity, the weights are distributed equally to the positive nodes. Furthermore, negative nodes are avoided, which are meaningless when dealing with the particle size as internal coordinate.

- For $N = 3$ it might still happen that complex nodes occur. To avoid this, a new option is provided that takes just the a-values (i.e., the diagonal elements) of the jacobi matrix as eigenvalues. In fact, this option is exactly valid in case the b-values are small. To choose the method, the new variable `'computeEigenValues'` is set to `'false'` in the *'qmomProperties'* file.
  Since with this option the nodes and weights are guessed rather than computed, the solver will display a warning when setting this variable to `'false'`. However, for the monodisperse case there should not be any deviations between the two options. Hence, we ran a growth simulation for initial droplets with a size distribution of a finite width. For that we chose a log-normal distribution with $\mu' = 5 \cdot 10^{-8}$ and $\sigma' = 0.1$ according to Eq. (B.8), and thus, the moments are calculated according to Eq. (B.9).
  For the chosen growth settings no countable deviations in the results occur in comparison with setting the variable to `'true'`, as it is shown in Fig. 5.3.

- Another issue was the precision for the moments solver itself, that can be adjusted in the *'fvSolution'* dictionary. In early tests the *'tolerance'* was set to `1e-10` and *'relTol'* was set to `1e-3`. It was found that setting the tolerances to `1e-32` and `1e-6`, respectively yielded much better results.
  (The proper choice of the precision is indicated by the output `'No Iterations xx'` in the log file of the simulation run, where `xx` should always be bigger than zero!).

- A further problem might arise in case of a high growth rate. In realistic applications the growth rate is mainly determined by the supersaturation that occurs inside a CPC. If the saturation ratio causes a high growth rate and the time step was chosen too large, stability problems might arise. This behavior becomes clear when considering Eq. 5.5.
  Nevertheless, the *qmomCloud* (in which our models are implemented) does not provide a time step refinement to avoid instabilities. Therefore we included a limitation of the growth rate with a limiting growth factor, that can be specified by the user in the *'qmomProperties'* file of the case.

- In cases where the set of moments becomes ill-conditioned, the higher order moments might become unstable (i.e., their magnitude might fluctuate). In order to be able to suppress these instabilities, a moment limitation was implemented that is detailed in the next paragraph.

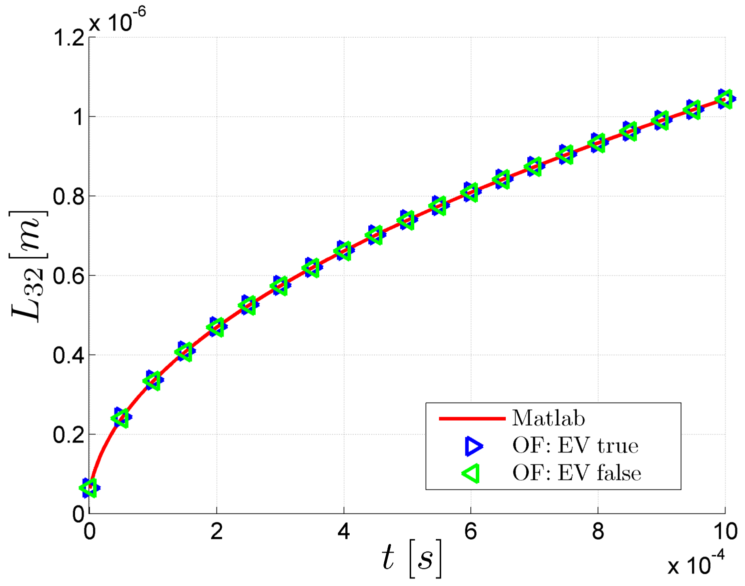


Figure 5.3: Time evolution of the Sauter mean diameter L_{32} for the growth of droplets with a log-normal size distribution with $\mu' = 5 \cdot 10^{-8}$ and $\sigma' = 0.1$, calculated with ML (red solid line), with OF using the correct eigenvalues by setting `'computeEigenValues'` to `'true'` (blue triangles) and with OF using guessed eigenvalues by setting `'computeEigenValues'` to `'false'` (green triangles). The initial moment set m_0 to m_5 was calculated according to Eq. (B.9). The simulation was run up to $1\,ms$ with a time step of $10^{-7}\,s$.

Limitation of the moments

In the course of our tests it turned out that in some cases the higher order moments become unstable due to numerical errors. This was especially observed in case of very low moment values.

Therefore, we have implemented a limitation of the moments that can be parametrized by the user. The limitation is based on the method of moments with interpolative closure (MOMIC). This method assumes that the higher order moments obey a potential law (see [8, Fig. 7.1]), and hence an inter- or extrapolation is easily possible. In the following this limitation of individual moments during droplet growth is demonstrated.

When applying the moment limitation algorithm, moments with order higher than a specified threshold are limited by the expected (i.e., extrapolated) moment m_k^{exp}:

$$m_k^{exp} = m_0 \left(\frac{m_{k-1}}{m_0} \right)^{\frac{k}{k-1}} \tag{5.7}$$

Specifically, when updating the moment set, which is calculated according to Eq. (5.5), the moment is bounded to the interval around m_k^{exp}. This interval is determined by the limitation factor $\overline{d}$.

$$m_k = \begin{cases} m_k & \text{for} \quad m_k^{exp}/\overline{d} \le m_k \le m_k^{exp} \cdot \overline{d} \\ m_k^{exp} & \text{else} \end{cases} \tag{5.8}$$

The limitation factor $\overline{d}$ can be specified by the user in the *qmomProperties* file by setting the variable `'momentLimitFactor'` in the `'momentTransportProps'`-sub dictionary. Moreover, the moment threshold (i.e., the order of the moment above the limitation is employed) can be specified by setting the variable `'limitMomentAbove'`. Moments with an order higher than this threshold will be limited, while for lower-order moments no limitation will be applied.

The effect of the moment limitation in a growth problem, with the same growth settings as before, is demonstrated next. We only use four moments and the moment limitation factor was set to $\overline{d} = 5$. The moment ID for the limitation was set to three, thus, only the third moment will be limited. The time step for this test was chosen to be $10^{-4}\,s$ and the simulation was run for 10 steps.

Tab. 5.6 shows the initial moments, as well as the moments after the first step. Since the moment m_3 is not in the limitation interval (m_3 is smaller than the lower limit $m_k^{exp}/\overline{d}$), it is corrected by increasing m_3 to m_3^{exp}. Also for the subsequent time steps the limitation algorithm yielded proper results.
In Fig. 5.4 the time evolution of the mean droplet size L_{32} is shown, and compared to the unlimited case. One can observe the larger mean size of the droplets after the first time step as a result of the limitation algorithm. However, after the second time step the mean size is reduced compared to the unlimited case due to the limitation. From the third time step onwards no limitation of the moments were made. Thus, the mean size grows in a similar manner as in the unlimited case.

Table 5.6: Moment limitation after the first time step: The table lists the moments at the beginning at $t = 0\,s$, the moments after the first step at $10^{-4}\,s$ and the corrected moments after the first step. The ID of the limited moment is 3, thus m_3 is corrected with a limitation factor of $\overline{d} = 5$.

	t = 0 s	t = 0.1 ms	t = 0.1 ms, corrected
m_0	1.000000E-00	1.000000E-00	1.000000E-00
m_1	5.000000E-08	5.366117E-06	5.366117E-06
m_2	2.500000E-15	5.341117E-13	5.341117E-13
m_3	1.250000E-22	3.999588E-20	3.903445E-19
	m_3^{exp}	$m_3^{exp} \cdot \overline{d}$	$m_3^{exp}/\overline{d}$
	3.903445E-19	1.951722E-18	7.806891E-20

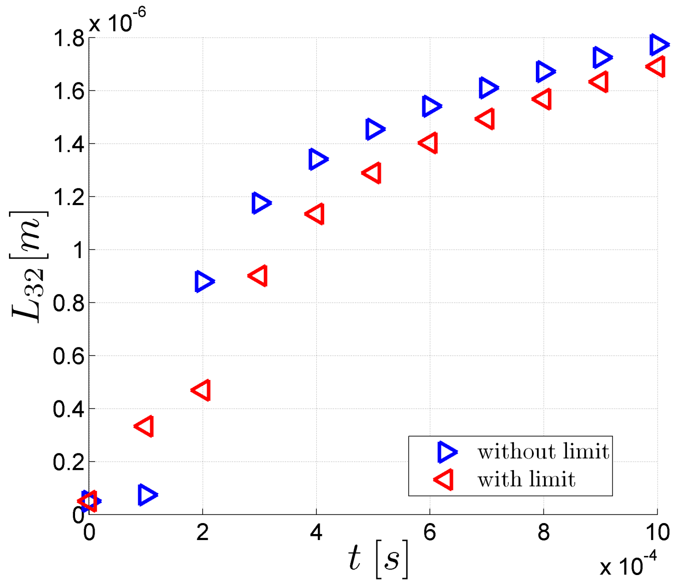


Figure 5.4: Moment limitation in case of pure growth affecting the time evolution of the Sauter mean diameter L_{32}. For the limitation a moment limitation factor of $\overline{d} = 5$ was used. The moment ID was set to 3, thus, the moment m_3, and therefore also L_{32} was corrected. The simulation was run with a time step of $10^{-4}\,s$ for 10 steps.

Note, for the monodisperse case the extrapolation formula Eq. (5.7) yields correct results since $m_k = d_p^k$ (see Eq. (5.4)). However, one should be aware that for distributions with finite width the accuracy of the extrapolation formula Eq. (5.7) depends on the distribution's width. This can be shown by considering a log-normal distribution, according to Eq. (B.8), with $\mu' = 4 \cdot 10^{-8}$ (acc. to a typical droplet size of $40\,nm$) and different values of σ'. In this case the extrapolation formula for the moment limitation Eq. (5.7) becomes

$$m_k^{exp} = \left[\exp\left((k-1)\ln(\mu') + \frac{(k-1)^2\sigma'^2}{2}\right)\right]^{\frac{k}{k-1}} \tag{5.9}$$

We now consider applying the limitation for the moment m_3. In Tab. 5.7 the values of the exact moment and of the extrapolated moments according to Eq. (5.9) are shown for different values of σ'. One can see that the accuracy of the method increases significantly with decreasing σ'. This means that the wider the distribution, the bigger is the error of the moment limitation.

Table 5.7: Moment limitation for a log-normal distribution function: The table shows the exact moment m_3 of a log-normal distribution with $\mu' = 4 \cdot 10^{-8}$ and different values of σ', calculated according to Eq. (B.9), in comparison to the extrapolated moment m_3^{exp} according to Eq. (5.9). Additionally, the relative error $rel.Error = |m_3 - m_3^{exp}|/m_3$ is listened.

	σ=1	σ=1/10	σ=1/100	σ=1/1000
m_3	5.761096E-21	1.003720E-22	6.694578E-23	6.428865E-23
m_3^{exp}	1.285474E-21	8.639096E-23	6.594909E-23	6.419229E-23
$rel.Error$	77.687%	13.929%	1.4888%	0.1499%

Finally, for pure growth, we want to test if the McGraw correction algorithm affects the growth behavior. The McGraw correction algorithm was tested based on Exercise 3.4 from [8]. In Sec. 5.2 it is shown that the correction algorithm works properly for this simple example. However, during our tests within the 3x3-test case we occasionally observed instabilities when activating the correction algorithm, especially for long term tests. Therefore, we have tested the growth example with a much larger time step of $\Delta t = 10^{-5}\,s$. The simulation was run up to $1\,s$ with different settings of the variables `'useCorrector'` and `'computeEigenValues'`.

When setting both, `'useCorrector'` and `'computeEigenValues'` to `'true'`, the higher order moments (m_3, m_4 and m_5) became unstable as shown in Fig. 5.7. Hence, the combination of these two settings is not recommended for pure growth problems.
When deactivating the eigenvalue computation (i.e., setting `'computeEigenValues = true'`), the simulation ran without instabilities up to $1\,s$. However, the growth behavior changes when activating the "corrector". As shown in Fig. 5.5 (a), the droplets growth is reduced compared to the cases where the correction algorithm was switched off. Moreover, in 5.5 (b) it can be seen that turning off the eigenvalue computation does not change the growth behavior.

Another finding is that when calculating the mean droplet size from different moments. As shown in Fig. 5.5 (a) L_{10} is different from L_{21} and L_{32} if the correction algorithm is activated. Since we deal with a monodisperse droplet size distribution all three mean

droplet sizes must yield the same. When deactivating the ”corrector”, all three mean sizes show perfect overlapping, as it should be (see Fig. 5.5 (b)).
This example has shown that the McGraw correction algorithm might fail, and thus, we recommend to avoid the usage of this option. In further tests we will set the `'useCorrector'` to `'false'`.

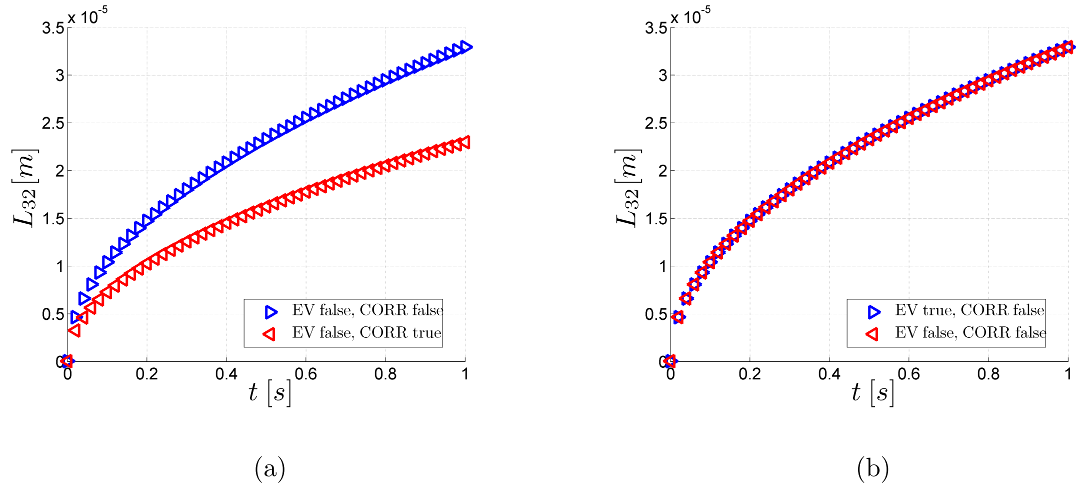


Figure 5.5: Effect of the McGraw correction algorithm on the growth behavior of droplets with an initial size of 50 nm: (a) effect of the variable `'useCorrector'` for `'computeEigenValues'` = `'false'`. (b) effect of the variable`'computeEigenValues'` for `'useCorrector'` = `'false'`. The simulation was run with a time step of $\Delta t = 10^{-5}\,s$ for 10^5 steps.

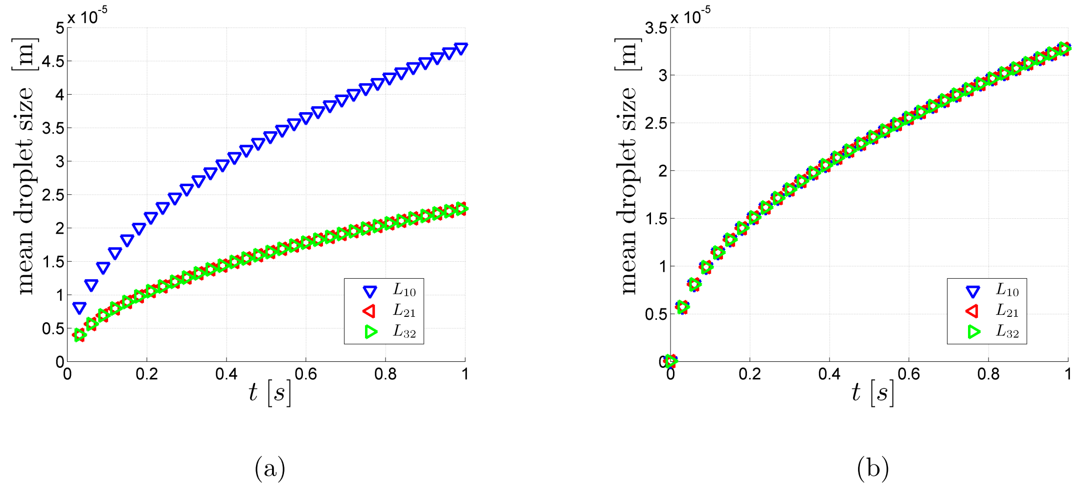


Figure 5.6: Effect of the McGraw correction algorithm on the growth behavior of droplets with an initial size of 50 nm with respect to the different mean sizes: The figures show the time evolution of three characteristic droplet diameters, i.e., L_{10}, L_{21} and L_{32}, when setting the variable`'useCorrector'` to `'true'` (a) and `'false'` (b). The variable`'computeEigenValues'` was set to `'false'`. The simulation was run with a time step of $\Delta t = 10^{-5}\,s$ for 10^5 steps.

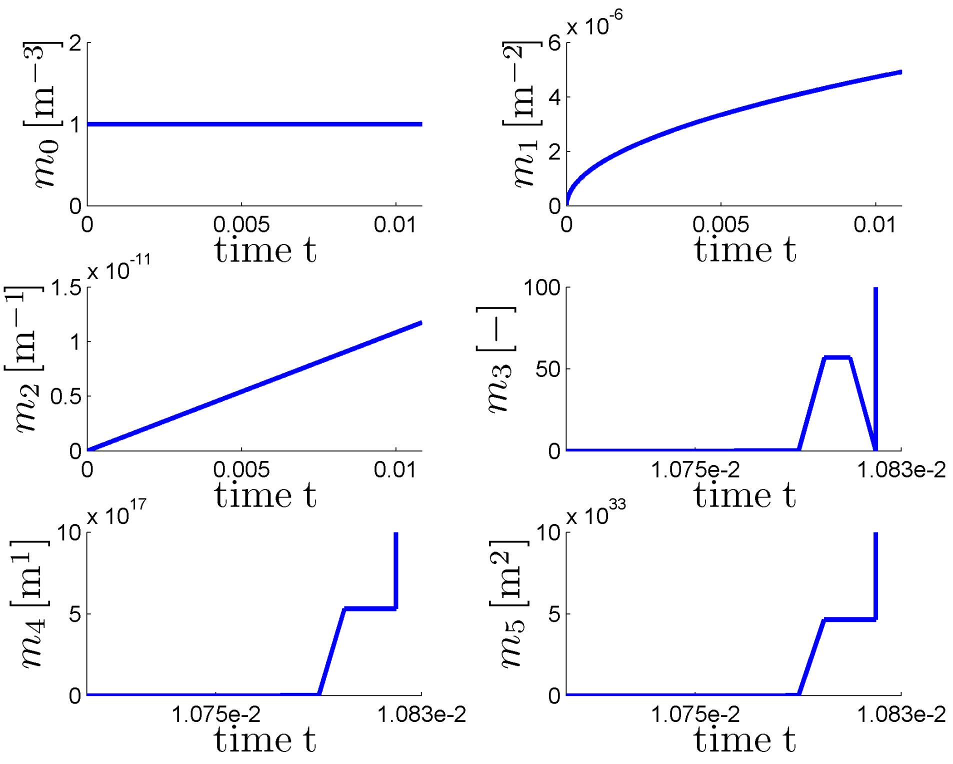


Figure 5.7: Effects of the McGraw correction algorithm on the growth behavior of droplets with an initial size of 50 nm with activated eigenvalue solver: The figure shows the time evolution of the moments up to a point where instabilities in the higher order moments occur. The simulation was run with `'useCorrector true'` in combination with `'computeEigenValues true'`, and a time step of $\Delta t = 10^{-5}\,s$. For the unstable moments m_3, m_4 and m_5 only the last few time steps are plotted in order to see the instabilities.

5.3.3 Case 3 - Nucleation

Since in a CPC high nucleation rates might exist, which might cause temporal jumps in the moment set, the robustness of the solver was tested for extreme nucleation conditions. Also, the results obtained with OF were compared to that of the ML code.
The moment equations for the pure nucleation case have the following closed form:

$$\frac{\partial m_k}{\partial t} = J_{het}\left(d_{d,init}\right)^k \tag{5.10}$$

The nucleation rate J_{het} is calculated according to the heterogeneous nucleation model in Sec. 3.5.4.
Before dealing with nucleation conditions like they occur in a CPC we want to investigate the general behavior of the heterogeneous nucleation model. Hence, for the following tests we used the moment set of Sec. 5.3.1 and the nucleation settings in the *'qmomProperties'* file were:

- $J^0_{het} = 30\,m^{-3}s^{-1}$
- $N_{prim} = 100\,m^{-3}$
- $d_p = 10^{-7}\,m$
- $\theta = 0°$

Note that the kinetic prefactor J^0_{het} was chosen to be very small (compared to real applications) in order to get a high nucleation time in the range of seconds. The maximum value of the particle number concentration N_{prim} is 100. The values for the working fluid properties were taken for n-butanol.
Note: to select n-butanol as the working fluid, the name for `'diffusingSpecies'` in the *transportProperties* dictionary must be set to `'C4H10On'`.

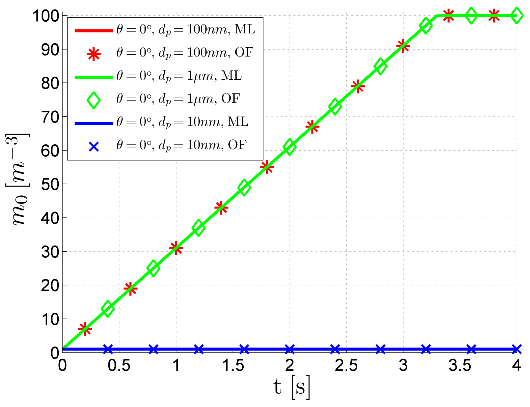
For n-butanol and the above settings, the Kelvin diameter d_k is about $34\,nm$. Since the primary particles are bigger than the Kelvin diameter, and because of the assumed perfect wetting, the nucleation rate $J \approx J^0_{het}$ (see App. A).

For this example (uniform initial size distribution) only the moment m_0 changes due to nucleation while the higher order moments remain nearly constant, since $d_{d,init}$ in Eq. (5.10) is very small. Contrary to that, in case of a zero initial moment set (see below) also the higher order moments change due to nucleation.

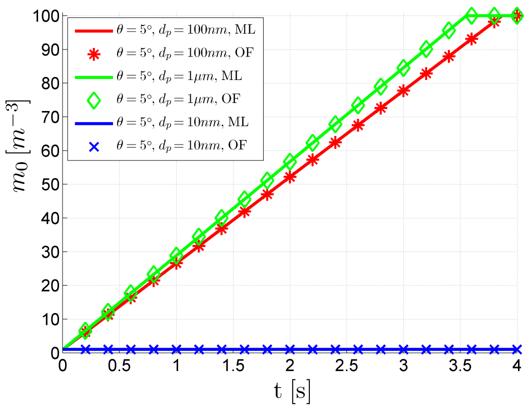
An important property of the heterogeneous nucleation model is that the nucleation rate is vanishingly small as long as the primary particles are smaller than the Kelvin diameter. For perfect wetting, the nucleation rate suddenly increases to J^0_{het} in case the seed particle size exceeds the Kelvin diameter. For higher values of the contact angle the nucleation rate changes more slowly (see Fig. A.3).

In order to test this behavior, the primary particle diameter was varied. Fig. 5.8 shows the time evolution of m_0 for three different primary particle sizes, $d_p = 10\,nm$, $d_p = 100\,nm$ and $d_p = 1\,\mu m$, for (a) perfect wetting, as well as (b) for a contact angle of $\theta = 5°$. In the case of $d_p = 10\,nm$, the particles are smaller than d_k. Hence, they do not have a sufficiently large size to activate the nucleation process and therefore m_0 remains constant.
For larger particles (i.e., $d_p = 100\,nm$ and $1\,\mu m$) m_0 grows with J^0_{het} for the case of perfect wetting. For $\theta = 5°$, it can be seen that the nucleation rate depends on the primary particle size. For $d_p = 1\,\mu m$ m_0 grows more quickly than for $d_p = 100\,nm$. This behavior agrees with the predictions of the heterogeneous nucleation theory.
Another observation is the correct limitation of the droplet concentration by the particle concentration (thus, only heterogeneous, but no homogeneous nucleation is predicted): in case the maximum particle concentration N_{prim} is obtained, m_0 remains constant and (heterogeneous) nucleation stops.

Furthermore, the impact of the contact angle on the nucleation rate was investigated. For that, the contact angle was varied from $\theta = 0°$ to $\theta = 20°$ in $5°$ steps. The primary particle size was set to $d_p = 100\,nm$.
The results are shown in Fig. 5.9. It can be seen, that the nucleation rate decreases with increasing contact angle, according to the predictions of the heterogeneous nucleation theory.



(a)



(b)

Figure 5.8: Heterogeneous nucleation of n-butanol vapor on seed particles with different sizes: time evolution of moment $m_0(t)$ for different primary particle sizes in case of (a) perfect wetting, and (b) with a contact angle of $\theta = 5°$. The solid lines show the calculated time evolution using the explicit Euler-method in ML, while the symbols show the results obtained with the *cpcFoamCompressible* solver in OF.

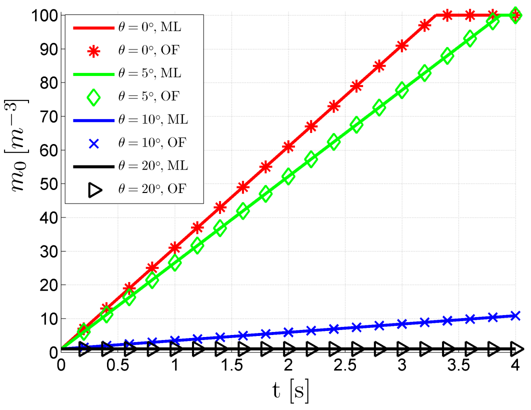


Figure 5.9: Heterogeneous nucleation on seed particles with a size of $100\,nm$ for different contact angles: time evolution of the moment $m_0(t)$ for contact angles of 0°, 5°, 10° and 20°. The solid lines display the calculated time evolution using the explicit Euler-method in ML, while the symbols show the results obtained with the *cpcFoamCompressible* solver in OF.

Note that for all test cases the results obtained with ML on the one hand and OF on the other hand show excellent agreement. Thus, the tests indicate the correct implementation and the proper behavior of the nucleation model conforming to the heterogeneous nucleation theory.

Nevertheless, in a CPC we have different conditions. On the one hand the disperse phase is formed by heterogeneous nucleation. Thus, the initial moments set equals zero, and suddenly becomes finite when the nucleation conditions are achieved. On the other hand we deal with higher number concentrations, and nucleation rates that might not be resolvable within one time step. Thus, more extreme nucleation conditions were considered next.
Specifically, the required fields, Y and T, were set to 1.1 and $319.15\,K$, respectively. Again, n-butanol was chosen as working fluid, and the required species parameters were computed according to the equations and parameters of Sec. C.
The nucleation parameters were modified to:

- $J^0_{het} = 7.8537 \cdot 10^{14}\,m^{-3}s^{-1}$
- $N_{prim} = 10^{10}\,m^{-3}$
- $d_p = 5 \cdot 10^{-8}\,m$
- $\theta = 0°$

So now we deal with $50\,nm$-particles and the number concentration of particles is $10^{10}\,m^{-3}$. The prefactor J^0_{het} was chosen to be $d_p^2 \pi 10^{29}\,m^{-3}s^{-1}$, according to [10]. The resulting value for J_{het} is $7.85397 \cdot 10^{14}\,m^{-3}s^{-1}$, and the value for the initial droplet diameter $d_{d,init}$ is $54.6\,nm$.
It is important to recognize that this kinetic prefactor J^0_{het} causes an extremely fast nucleation. This can lead to problems in the moment update procedure. Specifically, one

could update the moment using:

$$m_k^{t+1} = m_k^t + \Delta t \cdot J_{het} \cdot (d_{d,init})^{k-1} \tag{5.11}$$

Clearly, the time step has to be chosen sufficiently small to avoid an overshoot of the moment set due to nucleation. Consequently, a typical time step would be in the order of $10^{-9}\,s$. However, this would cause an enormous run time which is not practicable. Consequently, the integration algorithm was modified by limiting the nucleation rate such that an overshoot cannot occur.

For a first test we used a time step of $10^{-9}\,s$ and afterwards it was increased to $10^{-4}\,s$. In the case of $\Delta t = 10^{-9}\,s$ the moments grow linearly until $m_0 = N_{prim}$, as shown in Fig. 5.10. For $\Delta t = 10^{-4}\,s$, N_{prim} is already reached after the first time step, however, we end up with the same moment values as before.
This example shows that the behavior of the moments is stable despite the large time step and the fast nucleation process.

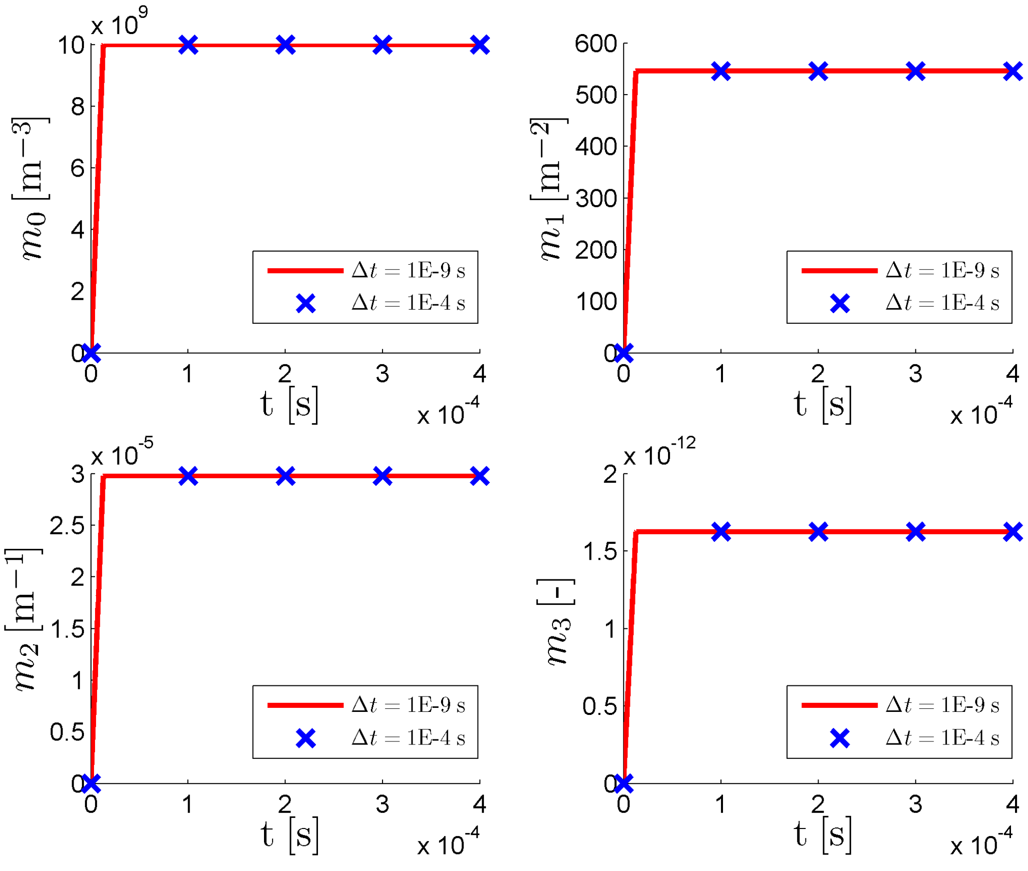


Figure 5.10: Formation of the disperse phase on $50\,nm$ seed particles with two different time steps: the time evolution of the moments calculated in OF is shown for a time step of $\Delta t = 10^{-9}\,s$ (red line) and for a time step of $\Delta t = 10^{-4}\,s$ (blue symbols). The nucleation rate was $J_{het} = 7.85397 \cdot 10^{14}\,m^{-3}s^{-1}$. In the former case of $\Delta t = 10^{-9}\,s$ the moments grow continuously up to the point where $m_0 = N_{prim}$, while in the latter case of $\Delta t = 10^{-4}\,s$ the formation of droplets is completed within one time step.

5.3.4 Case 4 - Nucleation and Growth

The moment transport equations that take into account nucleation and growth are

$$\frac{\partial m_k}{\partial t} = J\,(d_{d,init})^k + \left[2\,\frac{\rho}{\rho_l} D_v\, Sh\, \ln(1+B_m)\right] k\, m_{k-2} \tag{5.12}$$

The reliability of the two models (i.e., nucleation and growth) has already been tested separately for specific initial distributions (monodisperse, log-normal, uniform, etc.) of the droplet size. However, in a CPC the disperse phase is formed only at certain conditions (i.e., sufficient high supersaturation of the continuous phase), and generally, the formation takes place in the condenser. Before the fluid enters the condenser there is no disperse phase, and thus, all the moments are zero.
In order to demonstrate the reliability of our solver for these conditions, the following test case will use a zero initial moment set.
For heterogeneous nucelation we used the same settings as in the previous section, which were:

- $d_p = 5 \cdot 10^{-8}\, m$
- $J^0_{het} = 7.853975 \cdot 10^{14}\, m^{-3} s^{-1}$
- $N_{prim} = 10^{10}$
- $\theta = 0°$

With the chosen settings we get an initial nucleation rate of $J_{het} = 7.853975 \cdot 10^{14}\, m^{-3} s^{-1}$. The growth settings were the same as in Sec. 5.3.2, leading to an initial growth rate of $G = \frac{2}{d_{d,init}} \frac{\rho_g}{\rho_l}\, Sh\, D_v \ln(1+B_m) = 9.893177 \cdot 10^{-3} \frac{m}{s}$. Also for this combined test case a comparison between the ML code and the OF calculation was performed. The time step was set to $\Delta t = 10^{-4}\, s$ and the simulation time was 1 second.
In Fig. 5.11 the time evolution of the moments is shown. First, it can be seen that the ML and OF results show perfect agreement, indicating the correct implementation in OF. Furthermore, the combined model predicts a rapid increase of the 0-th moment due to nucleation, as well as the subsequent growth process as it should be. Note that m_0 remains constant while the higher order moments change due to growth, because only heterogeneous nucleation takes place.
In Fig. 5.12 the time evolution of the mean droplet sizes, L_{21} and L_{32}, is shown. Also here the formation jump at the first time step can be seen, whereas the mean size changes from zero to the value of the initial droplet diameter $d_{d,init}$, which is $54.83\, nm$ with the used settings. The two mean diameter, L_{21} and L_{32}, show perfect overlapping, which indicates the correctness of the formed moment set.

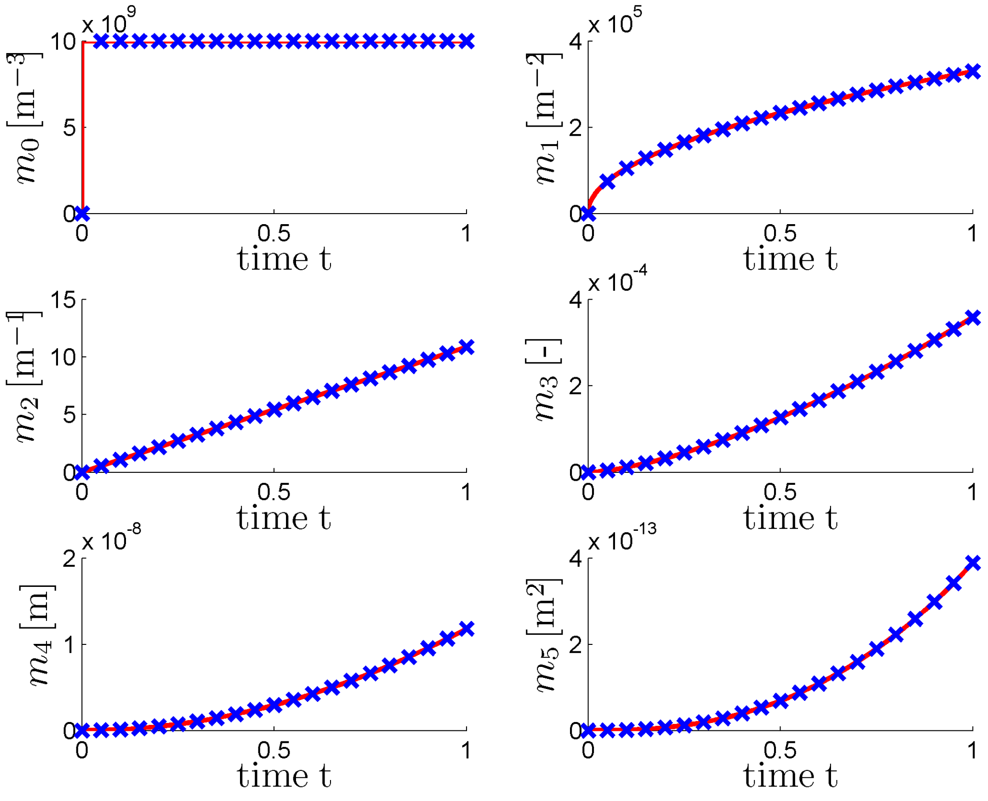


Figure 5.11: Nucleation and Growth on $50\,nm$ seed particles: time evolution of the moments m_0 to m_5 in ML (solid line) and OF (symbols) for a nucleation rate of $J_{het} = 7.853975 \cdot 10^{14}\,m^{-3}s^{-1}$ and an initial growth rate of $G = 9.893177 \cdot 10^{-3}\,\frac{m}{s}$.

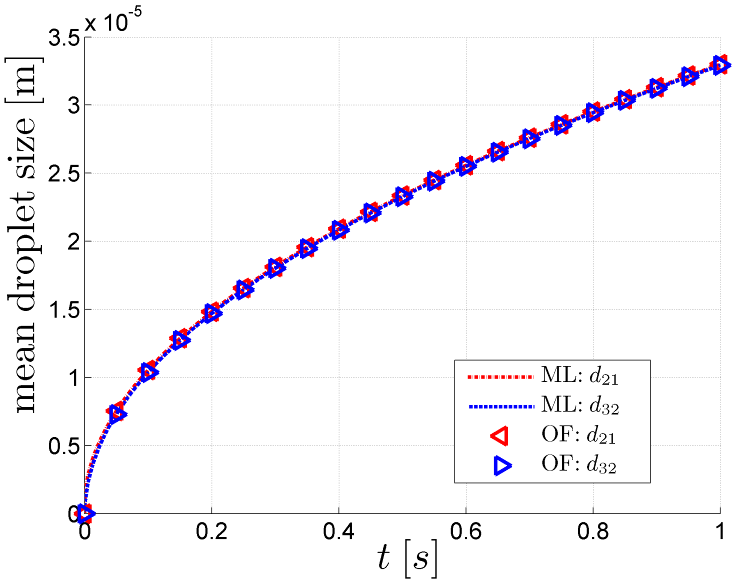
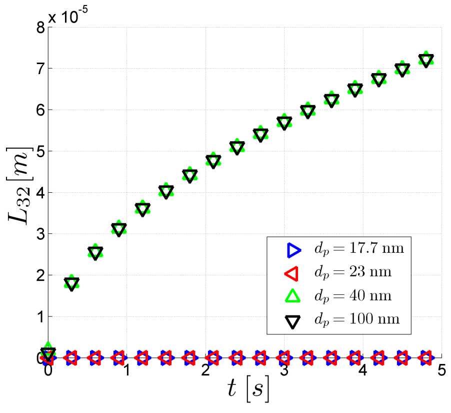
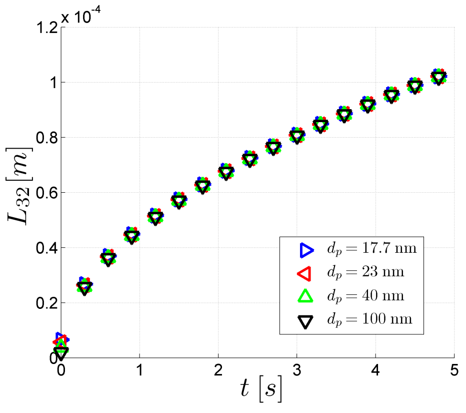


Figure 5.12: Nucleation and Growth on $50\,nm$ seed particles: time evolution of characteristic droplet diameters L_{21} and L_{32} in ML (lines) and OF (triangles) for a nucleation rate of $J_{het} = 7.853975 \cdot 10^{14}\,m^{-3}s^{-1}$ and an initial growth rate of $G = 9.893177 \cdot 10^{-3}\,\frac{m}{s}$.

Up to now we used a seed particle size of $50\,nm$, and thus, had always sufficient large particles for heterogeneous nucleation. Next, we consider nucleation and growth behavior for different primary particle diameter for a supersaturation of $S = 1.1$, which yields a Kelvin diameter of $33.5\,nm$. In Fig. 5.13 (a) the nucleation and growth behavior for different seed particle sizes is shown. The seed particles with sizes below d_k ($17.7\,nm$ and $23\,nm$) are not activated, since the heterogeneous nucleation factor f_g is finite and the resulting nucleation rate is zero. For the bigger particles ($40\,nm$ and $100\,nm$) droplet growth starts from their initial droplet sizes.
When increasing the supersaturation to $S = 1.2$, the growth rate changes from $G = \xi^{-1}5.4239 \cdot 10^{-10}\frac{m}{s}$ to $G = \xi^{-1}1.0793 \cdot 10^{-9}\frac{m}{s}$, and the Kelvin diameter becomes $d_k = 17.49\,nm$. As shown in Fig. 5.13 (b), under these conditions also the smaller particles activate, and droplets form.
Another aspect, shown in the figures, is that once the nucleation threshold is exceeded, the smaller particles grow faster than the bigger ones. Therefore, differences due to different initial droplet sizes become smaller. Hence, the long term behavior of the droplet sizes for different seed particles is quite similar. Since the growth rate is higher for a saturation ratio of 1.2, the final (mean) droplet size is larger compared to the case of $S = 1.1$.



(a)



(b)

Figure 5.13: Nucleation and growth of droplets for different seed particle sizes ($17.7\,nm$, $23\,nm$, $40\,nm$ and $100\,nm$) at a saturation ratio of (a) $S = 1.1$ and (b) $S = 1.2$. The growth rate in case of $S = 1.2$ is $G = \xi^{-1}1.07928 \cdot 10^{-9}\,m/s$, whereas for $S = 1.1$ it is $G = \xi^{-1}5.4236 \cdot 10^{-10}\,m/s$.

5.4 The Coupling Models

The last test within the 3x3-test geometry should prove the reliability of the coupling models as documented in Sec. 3.6. The calculation of the source terms (heat and mass) is performed in the files for the two relevant physical models, *'nucleation.C'* and *'simpleGrowth.C'*.
The coupling can be switched on in the *'constant/qmomProperties'* dictionary of the *'qmomTest_3x3'* tutorial case by activating the keyword `'couplingModel'`. Moreover, the transport equations for energy and mass of the continuous phase have to be activated in order to update the saturation ratio and the temperature. Hence, the keywords `'solveEnergyEqn'` and `'solveSpeciesEqn'` have to be activated in the *'system/controlDict'* dictionary.

For the following tests of the coupling model we want to track the quantitative changes in the affected quantities, Y, S and T. Their initial values were set in the *'0'*-directory of the case. For Y we used an initial value of 1.1, for T we set the initial value to $319.15K$. Further, we check the correct behavior of the solver, which means that the mass (and heat) transfer between the continuous and the disperse phase has to stop when S has decreased to one, i.e., the droplet growth process is terminated. Moreover, for pure nucleation the mass (and heat) transfer should also stop in case all seed particles have been activated, and a droplet has formed on their surface (i.e., $m_0 = N_{prim}$).

In order to compare the predicted change in the above mentioned quantities, it is useful to derive analytical expressions for the change of continuous-phase properties. Specifically, the mass transport equation without flow (see, Eq. (3.68)) is

$$\frac{\partial}{\partial t}\left(\rho Y\right) = -\frac{\dot{\mathcal{S}}_{coupl}}{y_{sat}} \tag{5.13}$$

Since ρ is constant for this test case, we obtain the change in Y during a short time interval Δt (in which the coupling source term is constant) as:

$$\Delta Y = -\frac{\dot{\mathcal{S}}_{coupl}}{y_{sat}\,\rho}\Delta t \tag{5.14}$$

In a similar manner the change in the enthalpy within a short time step can be calculated from Eq. (3.72) and Eq. (3.73) resulting in

$$\Delta h = -\frac{\dot{\mathcal{S}}_{coupl}\,h_v}{\rho}\Delta t \tag{5.15}$$

The change in the temperature within a short time intervall can be approximated by (under the assumption that c_p does not change significantly with the temperature, see

Eq. (3.74))

$$\Delta T \approx \frac{\Delta h}{c_p} \tag{5.16}$$

When knowing the change in the mass fraction and the temperature, the change in the supersaturation can be calculated, with the use of Eq. (3.71), as

$$\Delta S = \left(\frac{Y^{t+\Delta t}\, T^{t+\Delta t}}{p_v(T^{t+\Delta t})} - \frac{Y^t\, T^t}{p_v(T^t)} \right) \frac{\rho_{v,sat}^{ref}\, R_g}{MW_v} \tag{5.17}$$

Here, $\rho_{v,sat}^{ref}$ is the reference concentration of the saturated vapor at the evaporator patch of the domain, and is displayed in the log file of the case. The vapor pressure of the working fluid $p_v(T)$ is computed by the *cpcFoamCompressible* solver using Eq. (C.12). Note, because of the temperature dependence of S and p_v, and because of the temperature increase due to condensation, the reduction of the supersaturation S will be higher than the decrease of the mass fraction Y.

With the three formulas, Eq. (5.14), Eq. (5.17) and Eq. (5.16) we can reproduce the correct implementation of the coupling model by checking the changes in the respective field values within one time step.

5.4.1 Coupling - Nucleation

For the case of pure nucleation, the mass source is

$$\dot{\mathcal{S}}_{coupl}^{nuc} = \rho_l\, J_{het} \left(\frac{d_k^3 \pi}{6} \right) \tag{5.18}$$

As mentioned in Sec. 3.6, when dealing with typical seed particles for a CPC (i.e., a particle size in the range of $50\, nm$), the impact of nucleation on the vapor concentration is rather small. In order to see a countable effect, we enhance the effect of nucleation by assuming bigger seed particles. For the following test we used a seed particle size of $d_p = 50\, \mu m$, leading to a nucleation rate of $J = 7.853975 \cdot 10^{20}\, m^{-3} s^{-1}$. Furthermore, for the following coupling tests in case of pure nucleation, we used $N_{prim} = 10^{15}\, m^{-3}$, and the time step was set to $\Delta t = 10^{-6} s$. With the working fluid parameters (see Sec. C in the Appendix), we obtain a Kelvin diameter of $d_k = 33.5\, nm$ and a liquid density of $\rho_l = 791.49\, kg/m^3$. Thus, we obtain a mass source of $\dot{\mathcal{S}}_{coupl}^{nuc} = 12.1899 \frac{kg}{m^3 s}$.
With a mass fraction of $Y = 1.1$, a saturation mass fraction of $y_{sat} = 0.0933$, a total gas density of $\rho = 1.1\, kg/m^3$ and a time step of $\Delta t = 10^{-6} s$, using Eq. (5.14) we obtain the following change in the mass fraction within one time step as:

$$\Delta Y = -1.1877 \cdot 10^{-4} \qquad \Rightarrow Y^{t+\Delta t} = Y^t + \Delta Y = 1.099881$$

Furthermore, using Eq. (5.15) and with $h_v = 685087.4\,\frac{J}{kg}$ (acc. to Eq. (C.15)) we obtain a change in the enthalpy of $\Delta h = 8.06718\,\frac{J}{kg}$. The heat capacity was set to $c_p = 1005\,\frac{J}{kg}$ in the *thermophysicalProperties* dictionary, which is the value for air under typical conditions. Hence, with Eq. (5.16) we obtain

$$\Delta T = 7.6 \cdot 10^{-3}\,K \qquad \Rightarrow T^{t+\Delta t} = T^t + \Delta T = 319.1576\,K$$

At the chosen temperature of $T^t = 319.15\,K$ the vapor pressure is $p_v(T^t) = 3645.394\,Pa$. With $T^{t+\Delta t} = 319.1576\,K$ we get a vapor pressure of $p_v(T^{t+\Delta t}) = 3647.049\,Pa$. Thus, with $\rho_{v,sat}^{ref} = 1.018282 \cdot 10^{-1}\,\frac{kg}{m^3}$, $R_g = 8.314\,\frac{J}{mol\,K}$ and $MW_v = 74.123\,\frac{g}{mol}$ we obtain the change in the supersaturation within one time step according to Eq. (5.17) as

$$\Delta S = -5.889 \cdot 10^{-4} \qquad \Rightarrow S^{t+\Delta t} = S^t + \Delta S = 1.099411$$

Note, since J_{het} is zero if m_0 has reached N_{prim} (which means that a liquid film has formed on all particles) the mass and heat source for coupling is zero as well. However, if m_0 is lower than N_{prim}, the nucleation rate is non-zero and has a value of :

$$J_{het}^{red} = \frac{N_{prim} - m_0}{\Delta t} \tag{5.19}$$

Hence, the mass source is:

$$\dot{\mathcal{S}}_{coupl}^{nuc,red} = \dot{\mathcal{S}}_{coupl}^{nuc}\,\frac{J_{het}^{red}}{J_{het}} \tag{5.20}$$

Thus, after the first time step (at $t = 10^{-6}\,s$) $m_0 = 7.853975 \cdot 10^{14}\,m^{-3}$. After the next time step m_0 will reach N_{prim}. Hence, according to Eq. (5.19) the reduced nucleation rate is $J_{het}^{red} = 2.146025 \cdot 10^{20}\,m^{-3}s^{-1}$ leading to a reduced mass source of $\dot{\mathcal{S}}_{coupl}^{nuc,red} = 3.387098\frac{kg}{m^3 s}$. In addition, the enthalpy change is $\Delta h = 2.11\,\frac{J}{kg}$, and thus, we obtain the following changes for the next time step:

$$\Delta Y = -3.30016 \cdot 10^{-5} \qquad \Rightarrow Y^{t+\Delta t} = Y^t + \Delta Y = 1.099848$$

$$\Delta T = 2.01 \cdot 10^{-3}\,K \qquad \Rightarrow T^{t+\Delta t} = T^t + \Delta T = 3.191521\,K$$

With the updated vapor pressure $p_v(T^{t+\Delta t}) = 3.647507 \cdot 10^3$ and the new values for Y and T we get the change in the supersaturation:

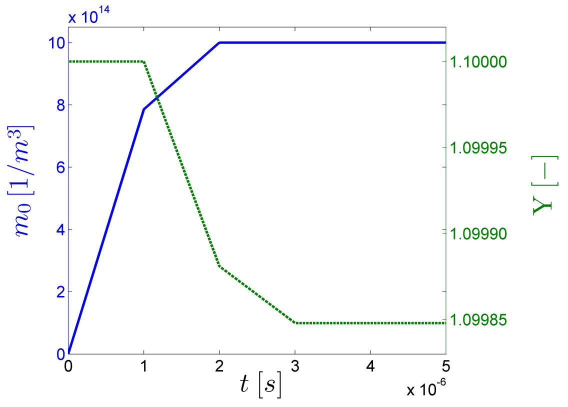
$$\Delta S = -1.637 \cdot 10^{-4} \qquad \Rightarrow S^{t+\Delta t} = S^t + \Delta S = 1.099247$$

With these settings OF simulations were run, and compared to the calculated values as indicated above. In Tab. 5.8 the values for m_0, Y, S and T from the OF simulation are presented. It can be seen that all the expected values for Y, T and S were obtained exactly.

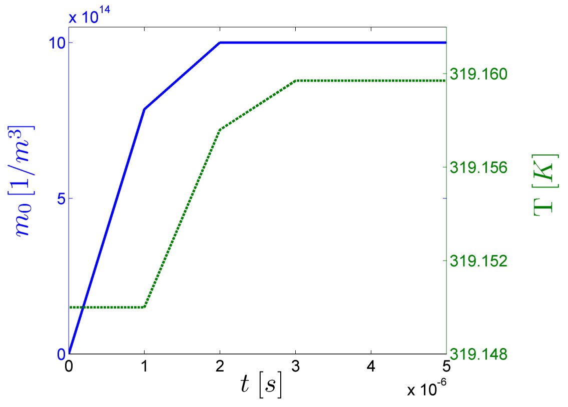
Table 5.8: Coupling in case of pure nucleation: The table lists the values for m_0, Y, T and S after the first three time steps.

	t = 1E-6 s	t = 2E-6 s	t = 3E-6 s
m_0	7.853975E+14	1.000000E+15	1.000000E+15
Y	1.100000E+00	1.099881E+00	1.099848E+00
T	3.191500E+00	3.191576E+02	3.191597E+02
S	1.100000E+00	1.099411E+00	1.099247E+00

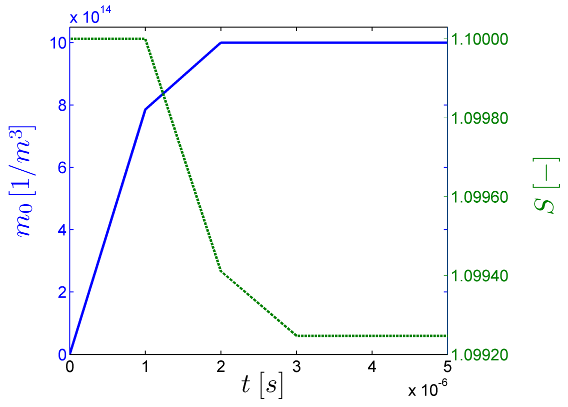
The progression of the values can also be seen in Fig. 5.14. After the formation of particles has stopped ($m_0 = N_{prim}$) the coupled quantities Y, S and T remain constant. Moreover, the figures show that the change in Y, T and S always occurs one step after the change in m_0 since the mass source is calculated using m_0. This delay of one time step is characteristic for numerical calculations. Nevertheless, the presented example showed the correct behavior of the implemented coupling model for nucleation.



(a)



(b)



(c)

Figure 5.14: Coupling in case of pure nucleation: time evolution of m_0 and (a) Y, (b) T, and (c) S. Droplets are nucleated on seed particles with $d_p = 50\,\mu m$ with an initial nucleation rate of $J = 7.853975 \cdot 10^{20}\,m^{-3}s^{-1}$. The simulation was run with a time step of $\Delta t = 10^{-6}\,s$ for 5 time steps.

5.4.2 Coupling - Growth

For growth, the mass source is given by

$$\dot{\mathcal{S}}_{coupl}^{growth} = \left[\rho \pi D_v Sh \ln(1 + B_m)\right] \, m_1 \tag{5.21}$$

For this test we want to use a number concentration of N_p monodisperse particles with a diameter of $d_p = 100\,nm$, resulting in an initial moment set that is given by

$$m_k = N_p \, d_p^k \tag{5.22}$$

Thus, with $N_p = 10^{15}\,m^{-3}$ we get $m_1 = 10^8\,m^{-2}$. The initial values for T and Y, as well as y_{sat}, ρ, c_p, MW_v, $\rho_{v,sat}^{ref}$ and h_v were the same as in the previous section. D_v was calculated according to Eq. (C.11), the Sherwood number was set to $Sh = 2$ and a spalding mass transfer number of $B_m = 1.2905 \cdot 10^{-2}$ was taken from the log file of the case (B_m is calculated by the solver according to the description in Sec. 3.5.1). With these values we can calculate the mass source to be $\dot{\mathcal{S}}_{coupl}^{growth} = 6.743377 \cdot 10^{1}\,\frac{kg}{m^3 s}$.

With a time step of $\Delta t = 10^{-6} s$ the expected change in the mass fraction according to Eq. (5.14) after the first time step is

$$\Delta Y = -6.5705 \cdot 10^{-4} \qquad \Rightarrow Y^{t+1} = Y^t + \Delta Y = 1.099343$$

For the change in the enthalpy we get $\Delta h = 41.1\,\frac{J}{kg}$, and thus, with Eq. (5.17) and Eq. (5.16) we obtain

$$\Delta T = 4.179 \cdot 10^{-2}\,K \qquad \Rightarrow T^{t+\Delta t} = T^t + \Delta T = 319.1918\,K$$

The vapor pressure at the initial temperature is again $p_v(T^t) = 3645.394\,Pa$, whereas vapor pressure with the updated temperature is $p_v(T^{t+\Delta t}) = 3654.507\,Pa$. Thus, we obtain the change in the supersaturation within one time step according to Eq. (5.17) as

$$\Delta S = -3.254 \cdot 10^{-3} \qquad \Rightarrow S^{t+\Delta t} = S^t + \Delta S = 1.096746$$

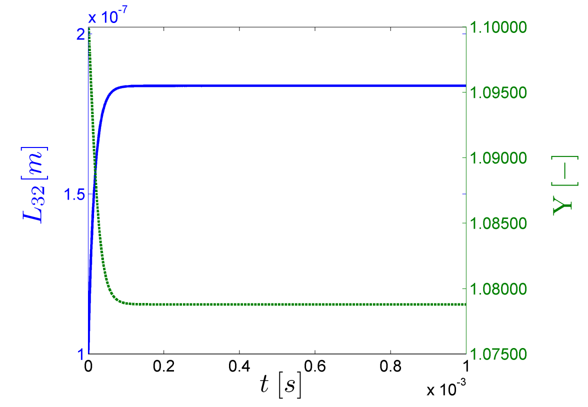
The simulation was run up to $1\,ms$. In Tab. 5.9 the values after the first two steps, and after the last time step are listened. At $t = 2 \cdot 10^{-6}\,s$ the expected values for Y, T and S were achieved.

The time evolution of the four listened quantities can be seen in Fig. 5.15. The continuous phase is depleted of the working fluid vapor (decrease of Y and S) during the growth of the droplets (increase of L_{32}) at the beginning of the simulation. When the supersaturation is reduced to one, the growth rate becomes zero, and therefore, the mass transfer to the

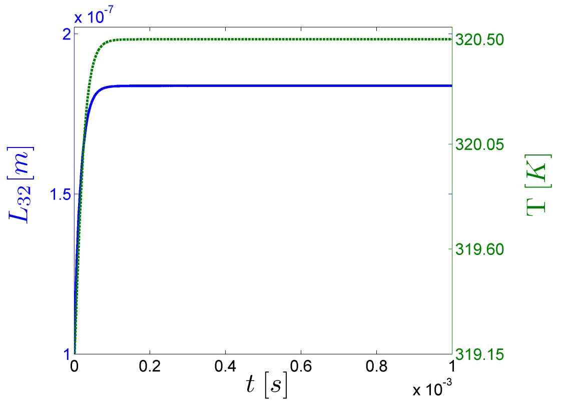
disperse phase stops. Hence, all the quantities remain constant.
Contrary to that, Fig. 5.16 shows the time evolution without coupling. There, the quantities Y, S, and T do not change while the droplets grow to an average size of about $10\,\mu m$. In the former case the depletion of the working fluid caused the stop of the droplet growth at $183.8\,nm$.

Table 5.9: Coupling for growth of droplets with an initial size of $100\,nm$: The values for L_{32}, Y, T and S after the first two steps, and after the last time step are listened.

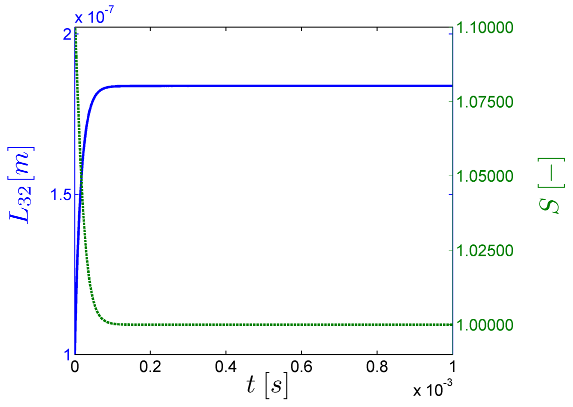
	t = 1E-6 s	t = 2E-6 s	t = 1E-3 s
L_{32}	1.048930E+07	1.095098E+07	1.837906E+07
Y	1.100000E+00	1.099343E+00	1.078794E+00
T	3.191500E+00	3.191918E+02	3.204971E+02
S	1.100000E+00	1.096746E+00	1.000000E+00



(a)

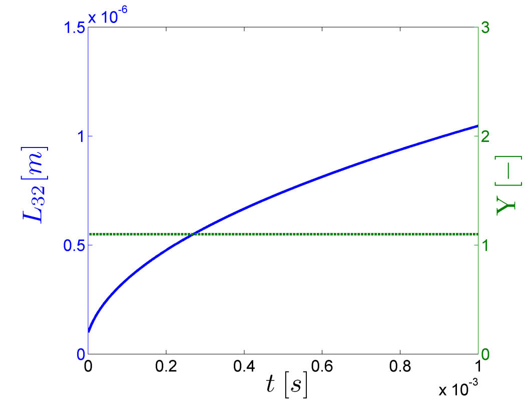


(b)

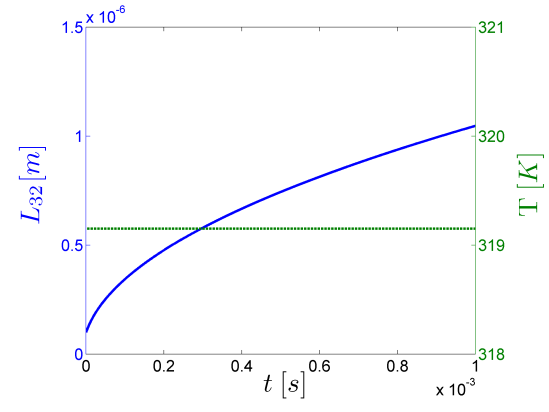


(c)

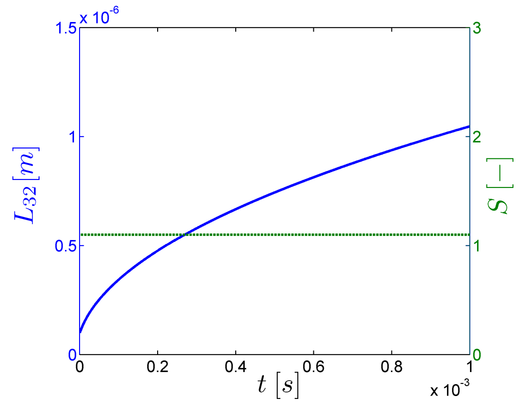
Figure 5.15: Coupling in case of pure droplet growth: time evolution of (a) Y, (b) T, and (c) S in comparison to that of L_{32} in case droplets with an initial size of $100\,nm$ grow with an initial rate of $G = 5.3538 \cdot 10^{-3}\,\frac{m}{s}$. The simulation was run with a time step of $\Delta t = 10^{-6}\,s$ up to $1\,ms$.



(a)



(b)



(c)

Figure 5.16: Pure droplet growth without coupling: time evolution of (a) Y, (b) T, and (c) S in comparison to that of L_{32} in case droplets with an initial size of $100\,nm$ grow with a rate of $G = 5.3538 \cdot 10^{-3}\,\frac{m}{s}$. There was no mass and heat transfer between the disperse and the continuous phase since the coupling was switched off. The simulation was run with a time step of $\Delta t = 10^{-6}\,s$ up to $1\,ms$.

5.4.3 Coupling - Nucleation and Growth

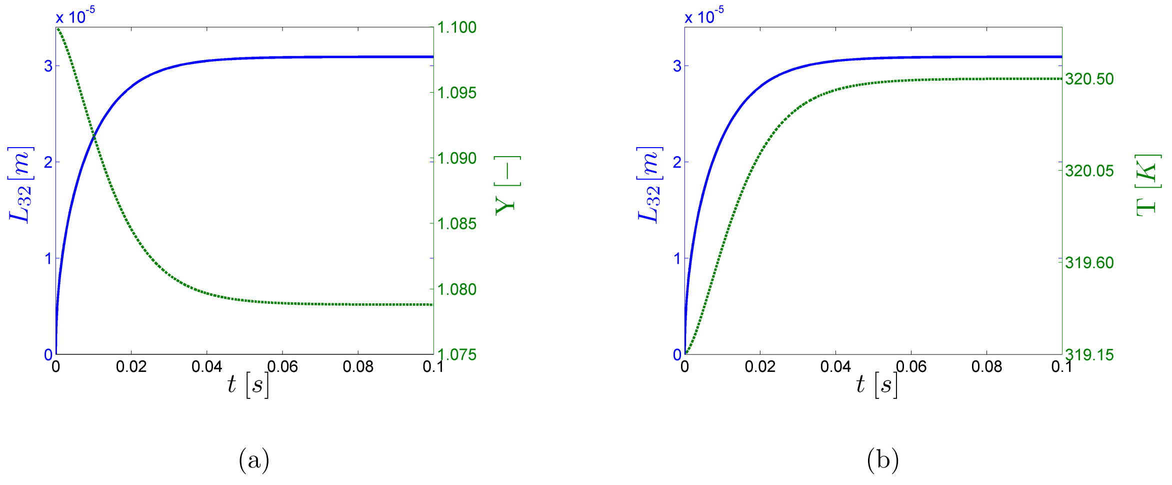
Finally, we want to test the coupling behavior when employing both, the nucleation and the growth model. With the same growth settings as before, a primary particle diameter of $50\,nm$ and a $N_{prim} = 10^{10}\,m^{-3}$ we obtain an initial nucleation rate of $J = 7.853975 \cdot 10^{14}\,m^{-3}s^{-1}$. With the used settings we obtain a Kelvin diameter of $d_k = 33.5\,nm$, and thus, the droplets start to grow from their initial droplet size of $d_{d,init} = 54.6\,nm$.

Using a time step of $\Delta t = 10^{-6}\,s$ the formation of droplets stops after 2 time steps, since all seed particles have activated the formation process. Therefore, in Fig. 5.17 the time evolution of Y, T and S is only compared with the mean droplet size L_{32}.

The figure shows the quick growth of the formed droplets at the beginning of the simulation, which causes the decrease of Y and S, indicating the depletion of the working

fluid. Since the growth rate depends on S, with increasing time the droplet growth decreases. At about $0.06\,s$ the supersaturation is close to one. Hence, the droplet growth stops and all quantities remain constant.
Contrary to the vapor concentration of the working fluid the temperature increases due to coupling until the mass (and heat) transfer between the continuous phase and the droplets stops.

In general, the tests have shown the correct implementation and functionality of the coupling models.



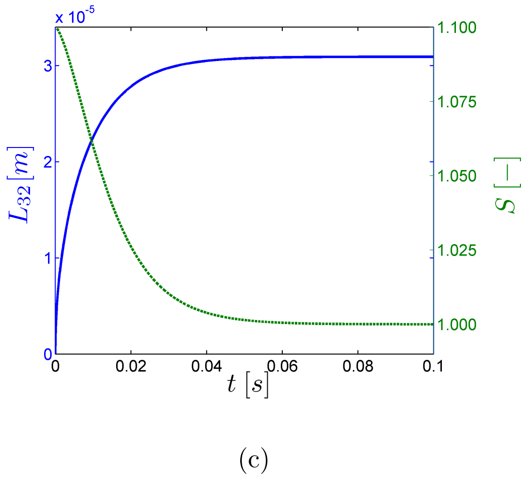


Figure 5.17: Coupling in case droplets are formed on seed particles with a size of $50\,nm$, and afterwards grow with an initial growth rate of $G = 9.8932 \cdot 10^{-3}\,\frac{m}{s}$. The figures show the time evolution of (a) Y, (b) T and (c) S in comparison to that of L_{32}. The simulation was run with a time step of $\Delta t = 10^{-6}\,s$ up to $100\,ms$.

6 The Pipe Flow Test Case

The reliability and performance of the two most important physical models, nucleation and growth, was tested in the simple 3x3-test geometry. These tests were done without considering spatial dependencies, which had the advantage of comparing the OpenFOAM-results with that of a Matlab code.

We now apply the *qmomCloud* library to a more realistic environment, namely a pipe flow situation typical for a CPC. In fact, the flow of a gas (i.e., air), partially saturated with vaporized working fluid, inside a cylindrical pipe is modeled. The pipe is divided into five sections with different boundary conditions for temperature and the (non-dimensional) vapor mass fraction. These changes of the boundary conditions lead to a certain supersaturation ratio in the pipe, such that nucleation and droplet growth takes place. A similar simulation of pipe flow, although without considering droplet nucleation and growth, was previously done by Tristan Reinisch (AVL) and Stefan Radl (TU Graz), and the same setup was used for the present study. The following modifications were necessary to activate the *qmomCloud* class in a simulation run:

- Initial Conditions
  In the *'0'* directory of the case the files for the new fields m_k (with k ranging between 0 and 5), as well as the corresponding weights and nodes were made. These files define the initial values of the fields at time zero, as well as the boundary conditions. All the initial field values were set to zero, since there is no disperse phase at the beginning of the simulation. All boundary conditions were set to *zeroGradient*.

- *qmomProperties* Dictionary
  In the *constant* directory of the case the *qmomProperties* dictionary was created to specify the settings for the moments transport equations.

- *controlDict* Dictionary
  The update of the disperse phase properties has to be activated in the *system/controlDict* dictionary of the case. At the top of the file, where the main switches for the solver are set, the following line was inserted to activate the solution for the disperse phase:

```
solveParticleEqns;    //activate to update particle cloud
```

- *fvSolutions* Dictionary
  In the *system/fvSolutions* dictionary of the case the finite volume solver for the moments has to be speficied by inserting the following code lines:

```
"(m)"
    {
        solver          PBiCG;
        preconditioner  diagonal;
        tolerance       1e-30;
        relTol          0;
        maxIter         5;
    }
```

- *'fvSchemes'* Dictionary
  Finally, in the *'system/fvSchemes'* dictionary the finite volume scheme for the moments must be added. For the advection term in the moment transport equations the following code line was inserted in the `'divSchemes'` sub dictionary:

```
div(phi,m)       bounded Gauss  upwind;
```

  For the diffusive term the following line was inserted in the `'laplacianSchemes'` sub dictionary

```
laplacian(Dp,m) Gauss linear corrected;
```

Note, here we use the `'upwind'` scheme for advection, which is a first order numerical scheme as it is used in Eq. (3.39) for advection. Hence, unrealizable moments should be avoided in case Eq. (3.41) is satisfied (see the description in Sec. 3.4.2).
The mentioned files for the pipe flow case are attached in the Appendix E.2.

6.1 Pipe Geometry

As already mentioned, the pipe flow case models a flow through a pipe with five sections, called inlet, evaporator, insulation, condenser and outlet. Since this problem is radially symmetric, it can be reduced to an axisymmetric (two-dimensional) computational domain, as shown in Fig. 6.1. Note, that OpenFOAM requires the specification of a three-dimensional domain in a Cartesian coordinate system. Consequently, a wedge-shaped geometry must be used to model an axisymmetric computational domain.

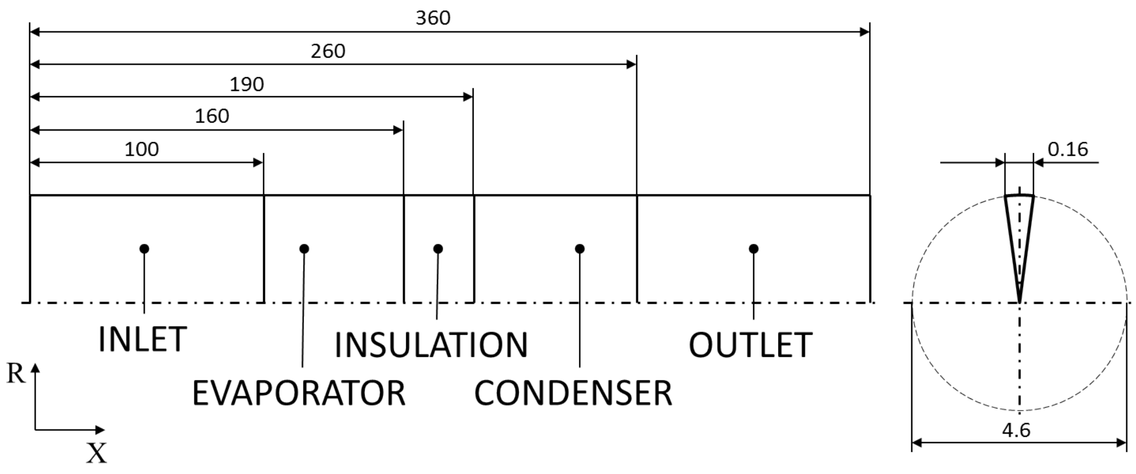


Figure 6.1: Schematic drawing of the domain in which the pipe flow is modeled. Dimensions are in *mm*.

The geometry is defined in the *'constant/polyMesh/blockMeshDict'* dictionary of the case (see Appendix E.2). The domain was spatially discretized as follows:

- radial discretization: the R-direction (which corresponds to the z-direction in the Cartesian coordinate system used by OF) was divided into 30 cells with a grading of 0.5. Thus, the radial cell size is refined in a way such that the cells at the pipe wall have half the size of the cells at the center.

- axial discretization: the X-direction for the sections inlet and outlet was divided into 100 cells (per section), whereas for the sections evaporator, insulation, and condenser 200 cells were used (per section).

Hence, the domain is divided into a mesh with 24000 wedge-shaped cells.

6.2 Settings

6.2.1 Initial and Boundary Conditions

The initial and boundary conditions were specified in the *'0'*-directory of the case. Since all simulations yielded a steady-state solution, initial conditions are only relevant to guarantee a stable simulation run. In order to save run time, however, the calculated steady-state solutions were taken as the initial conditions for all further simulations.
In the following, the most important boundary conditions used in the simulation runs are presented. Furthermore, key parameters of the working fluid are summarized.

- Temperature T
  One of the fields of main importance is the temperature field, since the supersaturation in the condenser is determined by the temperature difference between evaporator and condenser. For the walls up to the insulation section we used a wall temperature of $311.45K$. From the condenser section downstream the wall temperature was set to $304.85K$. These conditions are identical to ones used by [10].
  In the rest of the domain the initial cell temperatures were set according to Fig. 6.2. The temperature crossover at the beginning of the condenser is due to the flow in X-direction.

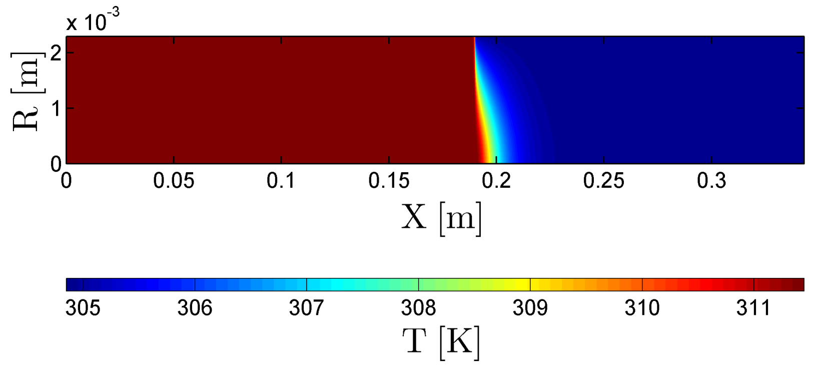


Figure 6.2: Initial temperature profile inside the pipe. Up to the condenser (at $X = 0.19\,m$) the temperature was set to $38.3°C$, from the condenser downstream it was set to $31.7°C$. The domain is compressed in the X-direction by a factor of 40.

- Velocity u

  Since a laminar flow prevails inside the pipe, the velocity profile is given by

  $$u(R) = 2u_{av}\left(1 - \left(\frac{R}{R_W}\right)^2\right) \tag{6.1}$$

  with R_W being the inner radius of the pipe. The average velocity u_{av} can be computed from the volumetric flow rate $\dot{V}$ and the cross sectional area of the pipe, i.e., $u_{av} = \dot{V}/(R_W^2\pi)$.

  In OF the velocity field was generated by solving the Navier-Stokes equations directly, which yields the velocity profile discussed in the last paragraph. An average velocity of $u_{av} = 0.125\,m/s$ was used. Note, since the velocity profile will not change the switch for the Navier-Stokes equation was deactivated in the *controlDict* file by commenting out the key word `solveFluidFlow;`.

- Vapor mass fraction Y

  The spatial distribution of the (non-dimensional) vapor mass fraction is shown in Fig. 6.3. It can be seen that in the inlet section Y is zero. The working fluid is evaporated at the walls of the evaporator patch. At this patch the reference vapor mass fraction is defined, thus at the evaporator walls Y is exactly unit. At the walls of the insulation patch no working fluid is evaporated (i.e., a zero-gradient boundary condition for Y is employed). However, the temperature is still the same as at the evaporator walls. At the condenser, a fixed value for Y, corresponding to the (non-dimensional) saturation vapor mass fraction at the condenser temperature, has been specified.

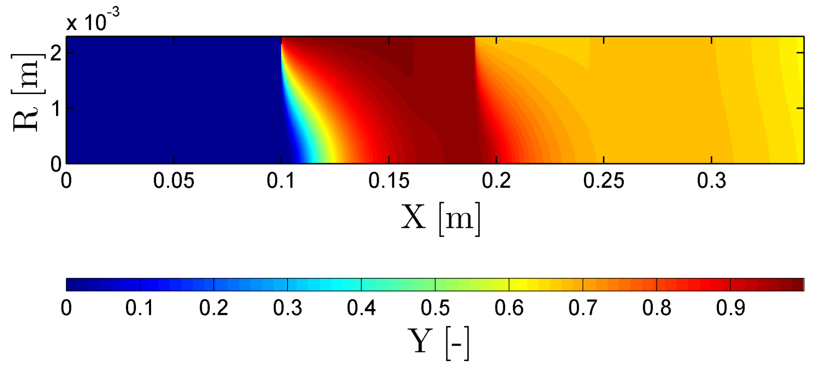


Figure 6.3: Vapor mass fraction for the pipe flow case. The working fluid is evaporated at the walls of the evaporator patch ($0.1\,m < X < 0.16\,m$). The domain is compressed in the X-direction by a factor of 40.

6.2.2 Working Fluid and Gas Properties

Following the report of Mamakos [10] we chose n-Butanol as the working fluid for the pipe flow case. The properties of the working fluid are specified in the *transportProperties* dictionary of the simulation case, and summarized in Sec. C of the Appendix.

Furthermore, the density of the gas (i.e., the mixture of the exhaust gas and the vaporized working fluid) is set to $\rho = 1.1614\,kg/m^3$ (see the settings in the *thermophysicalProperties* dictionary). Thus, the density does not change with temperature and pressure. This is consistent with the previous analysis of Mamakos [10].

6.2.3 Settings for the *qmomCloud*

In the pipe flow case we apply the nucleation and growth model, as well as the coupling model. Based on the experience gained in the 3x3-tests case, the following settings for the *qmomCloud* were used:

- Number of Moments

  For the pipe flow case the number of moments was chosen to be four. The reason for modeling the disperse phase with only four instead of six moments was that instabilities tend to occur in the moments of order higher than three. However, since the desired result for the disperse phase is the droplet size, four moments still provide enough information to compute the Sauter mean diameter. Moreover, when tracking four instead of six moments, the run time is significantly lower, since a smaller number of transport equations has to be solved. Of course, the accuracy of the quadrature approximation is lowered when dealing with only two nodes and weights. Hence, the chosen number of moments is a trade-off between stability and accuracy.

- Product-difference Settings
  According to the 3x3-test results, the following settings for the *PD* solver were used:

```
productDifferenceProps
{
    verbose             false;
    iter_max            10;
    useCorrector        false;
    roundPrecision      1e-4;
    minConcentration    1e5;
    computeEigenValues  true;
    useEqualWeights     false;
};
```

- Nucleation Parameters
  According to the maximum primary particle concentration [10, p. 31] we want to deal with a maximum of $N_{prim} = 10^{10}\,m^{-3}$ seed particles with $d_p = 40\,nm$. Furthermore, the nucleation prefactor was calculated according to Eq. (3.55), and perfect wetting was assumed. Hence the nucleation settings are:

```
nucleationProps
{
    verbose      false;
    SName        "S";
    theta        0;
    dParticle    40e-9;
    J0           5.02655e14;
    NPrim        1e10;

    cellId2Report 15001;    //id of the cell to report nuc. props
}
```

- Growth Parameters

```
simpleGrowthProps
{
    verbose      false;
    useLocalG;
    G0           1.0;
    r            -1;
    useQmom      true;
    SName        "S";
    YName        "Y";
    RhoGName     "rho";
    DiffName     "diffEff";
    limitGrowthFactor 1e8;
}
```

- Moment transport Settings
  The diffusion model according Stokes-Einstein diffusion model was deactivated and the constant droplet diffusion coefficient was set to zero. Furthermore, the moment limitation for the moment m_3 was activated with a limitation factor of five. This limitation factor will not cause a limitation of the moment set, except for rare situations in which numerical instabilities might lead to an unphysical increase of m_3.

```
momentTransportProps
{
    verbose                true;
    fieldName              "m";
    Dp0                    0;
    limitMomentAbove       3;
    momentLimitFactor      5.0;

}
```

6.3 Pipe Flow without Disperse Phase

First, we present the results of the pipe flow case without applying the *qmomCloud* library. Thus, only the solution of the continuous phase is obtained. The most interesting part of the pipe is the condenser, since there, the nucleation takes place. Therefore, in the following we will only present the results at this patch. The key result is the saturation ratio S, since the nucleation only takes place at location of sufficiently high supersaturation. By knowing the temperature field, the Kelvin diameter can be calculated according to Eq. (3.50). The Kelvin diameter indicates (approximately) the necessary seed particle size for heterogeneous nucleation to occur.

When running the simulation for the pipe flow case without employing the *qmomCloud*, a steady-state solution is obtained after approximately about $0.3\,s$. Nevertheless, in order to deal with the same situation as in the next section, we ran the simulation up to $1.2\,s$. In Fig. 6.4 the profile of the supersaturation in the condenser is displayed. The highest supersaturation is located near the cylinder axis, which is typical for a CPC and also a finding of previous studies (see [10] and [11]).
Fig. 6.5 (a) shows the profile of S along the cylinder axis, as well as that of the Kelvin diameter. The maximum value of the saturation ratio is 1.1849 at $X = 0.2049\,m$.
Fig. 6.5 (b) shows the radial profile of S at the X-position of the maximum supersaturation, as well as the Kelvin diameter. At the position of the maximum supersaturation the Kelvin diameter is $20.7\,nm$. Thus, particles that induce heterogeneous nucleation have to be larger than this critical size. For seed particles with a size of $40\,nm$ heterogeneous nucleation will occur within a radius of $R \approx 1.4\,mm$ (see Fig. 6.5 (b)). Therefore, when applying the *qmomCloud* with seed particles of $40\,nm$, the formation of the disperse phase within this radius can be expected.

Note, in [10], where the saturation ratio profile was calculated by solving the Graetz problem (see [9]), the maximum value of S was 1.2177 at an axial position of $1.4243 \cdot 10^{-2}\,m$, measured from the condenser inlet. This corresponds to an axial position of $X = 0.2042\,m$ in the pipe flow case. Thus, compared to the results in [10], the obtained maximum supersaturation in our case is slightly lower, while its position is well captured.

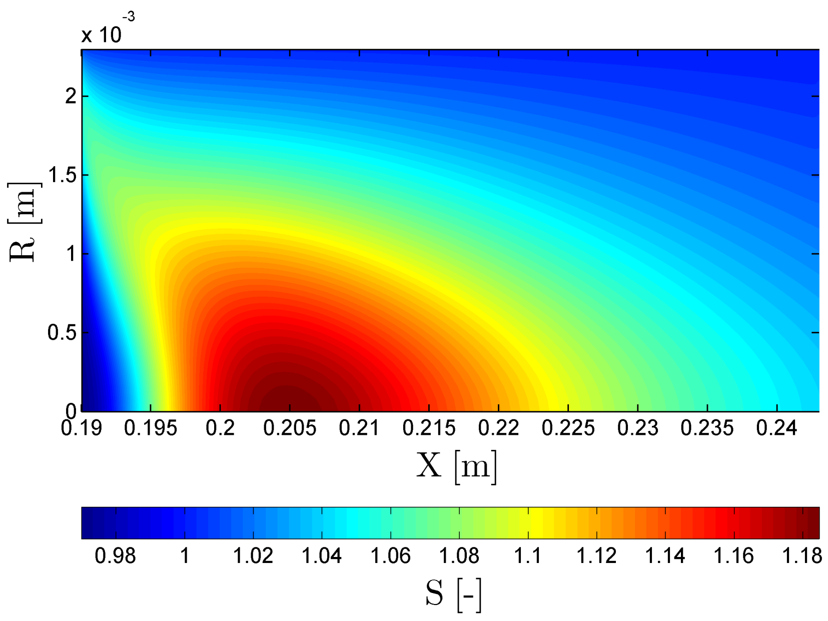
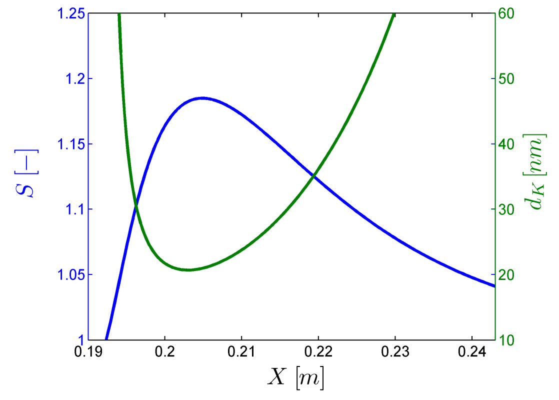
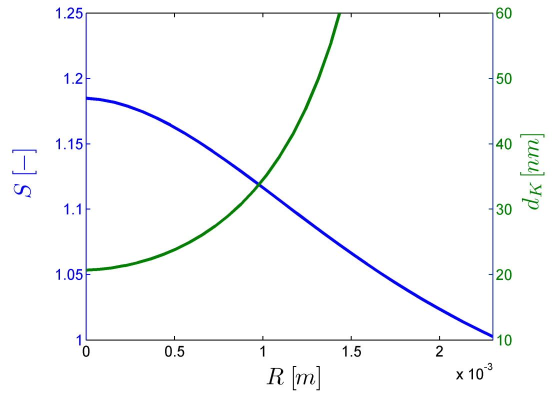


Figure 6.4: Two-dimensional supersaturation profile without disperse phase in the condenser, $t = 1.2\,s$



(a)



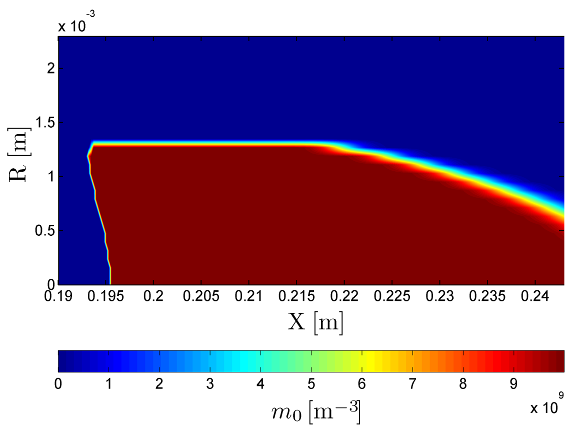
(b)

Figure 6.5: Profiles of supersaturation and Kelvin diameter without disperse phase, data has been sampled from 6.4 along (a) the X-axis, and (b) along the R-axis at the axial position of the highest supersaturation $X = 0.2049\,m$.

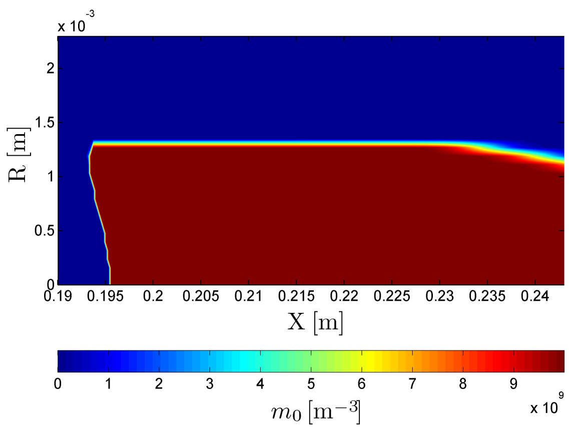
6.4 Pipe Flow with Disperse Phase - no Coupling

We now focus on a simulation run that includes the modeling of the disperse phase, but not coupling. Thus, the effect of the droplet formation on the continuous phase is neglected, and the identical temperature, vapor mass fraction, and supersaturation profiles were observed. In the *controlDict* dictionary the *qmomCoud* library was activated by including the variable `'solveParticleEqns'`.
The tests showed that a steady-state solution for the moment set was obtained after about $t = 0.8\,s$. Hence, all following simulations were terminated after $t = 1.2\,s$, in order to be on the safety side. The time step was set to $\Delta t = 10^{-6}\,s$.

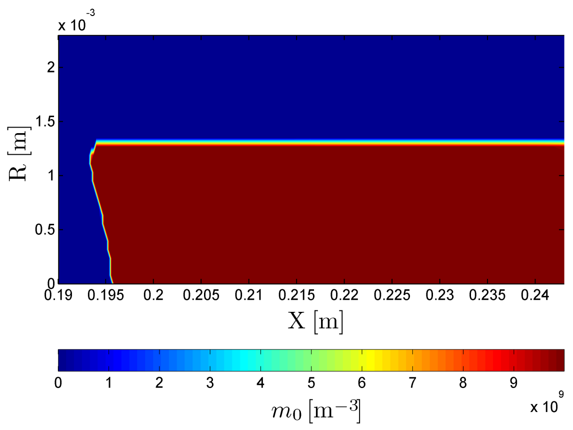
In Fig. 6.6 the time evolution of the moment m_0 is plotted, indicating that already after $t = 0.1\,s$ (Fig. 6.6 (a)) droplets are formed in regions of high supersaturation. The dark red region indicates that all of the seed particles ($10^{10}\,m^{-3}$) have activated the droplet formation, while the dark blue color exposes regions void of droplets. The interface between the two regions is quite sharp, especially at the left-hand side. At the right-hand side the interface is more diffuse due to the numerical diffusion inherent to the convection scheme.
The disperse phase moves with the gas into the positive X-direction (see (Fig. 6.6 (b), (c)). The downstream interface of the disperse phase moves into the positive X-direction, while the upstream interface does not move since new droplets are continuously formed. As expected from the results in the former section, the disperse phase is formed within about $R = 1.4\,mm$ and does not move in radial direction (because diffusion of the droplets was deactivated). After $t = 0.5\,s$ the droplet cloud has moved downstream to the end of the condenser section, and thus, a steady-state solution for the moment m_0 is obtained.



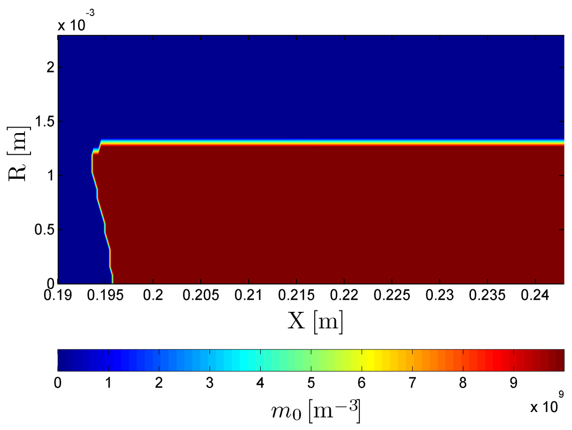
(a)



(b)



(c)



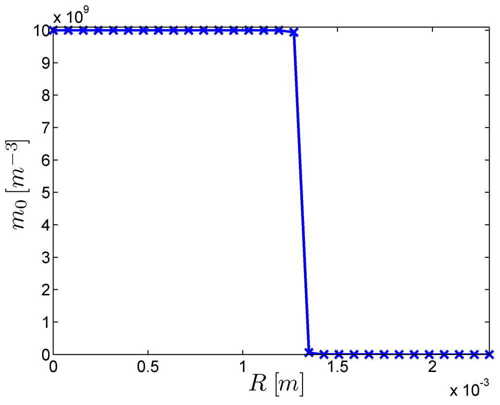
(d)

Figure 6.6: Time evolution of the moment m_0 in the condenser without coupling for (a) $t = 0.1\,s$, (b) $t = 0.2\,s$, (c) $t = 0.3\,s$, and (d) $t = 1.2\,s$.

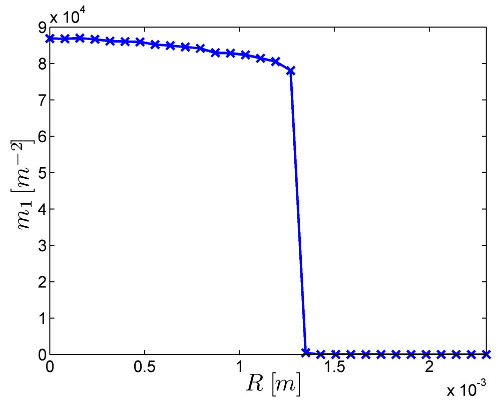
The formation and growth of the droplets takes place in the condenser region of the pipe where the supersaturation is larger than unit. Thus, when the droplets have left the condenser, the growth process is nearly completed. This means that the droplets' final size (i.e., the size detected by the light scattering device) is close to the droplet size at the end of the condenser section. Therefore, in the following the results are plotted along the radial axis at the condenser outlet (i.e., at $X = 0.243\,mm$), in accordance with Mamakos [10, Fig. 14].

In Fig. 6.7 the steady-state values of the moment set at the condenser outlet are shown. The moment m_0 (Fig. 6.7 (a)) is close to $10^{10}\,m^{-3}$ up to $R \approx 1.4\,mm$ and zero above, since m_0 is affected by nucleation only. This indicates that between a radial position of $R \approx 1.4\,mm$ and the pipe wall no droplets are present.

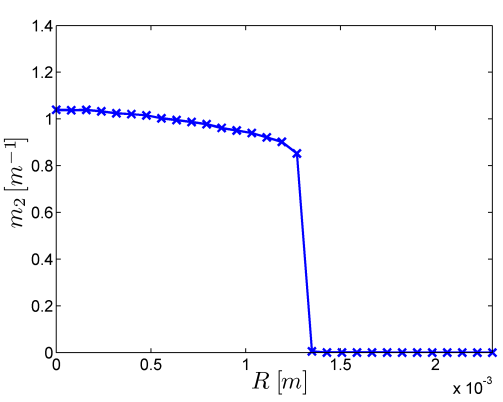
The moments m_1, m_2 and m_3 (Fig. 6.7 (b), (c) and (d)) have changed due to growth after the formation was completed. Thus, they show higher values near the cylinder axis because the supersaturation is higher at lower radial positions.



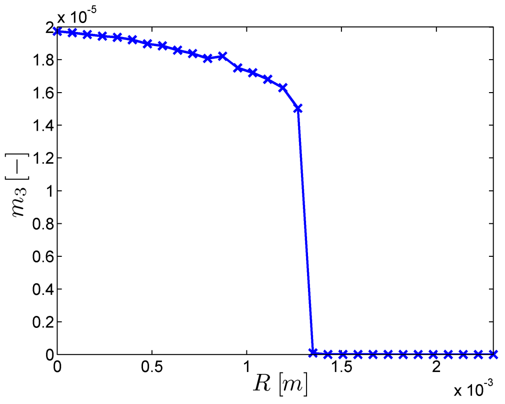
(a)



(b)



(c)



(d)

Figure 6.7: Moment set at the end of the condenser without coupling: steady-state values after $t = 1.2\,s$ of the moments (a) m_0, (b) m_1, (c) m_2, and (d) m_3 at the exit of the condenser, i.e., $X = 0.243\,mm$, $\Delta t = 10^{-6}\,s$.

The quantity of interest is the droplet size at the end of the condenser. As already mentioned, the mean droplet size can be obtained in various ways. When dealing with four moments the mean droplet sizes L_{10}, L_{21} and L_{32} can be calculated. The Sauter mean diameter L_{32} is the ratio of the total droplet volume and the droplet surface area. Hence, L_{32} has a sensible physical meaning, and can be used to estimate the droplet surface area once the droplet volume is known. In contrast, L_{10} is the arithmetic droplet size which is of lower significance, since small particles are weighted strongly. The characteristic diameter L_{21} is included as well in the presentation for completeness.

In Fig. 6.8 these three diameters are plotted at the condenser outlet. It can be seen that the three different mean diameters show big differences, which indicates that the final droplet size distribution has a polydisperse character. The Sauter mean diameter L_{32} is about twice the mean droplet size L_{10}. When considering the Sauter mean diameter,

the nucleated droplets grow to a mean size of about $19\,\mu m$, while L_{21} is approximately $12\,\mu m$. When considering L_{10}, a mean droplet size of about $8\,\mu m$ is obtained at the condenser outlet. This value is comparable to the results of Mamakos [10, Fig. 14] where the droplets grow up to a size of about $8\,\mu m$ in case of $40\,nm$ seed particles. (Note, in [10] the growth of monodisperse droplets was calculated.)
Nevertheless, in [10] the supersaturation ratio within the condenser was higher, and therefore, also the mean droplet size should be somewhat higher. Moreover, the particles get activated up to a radial position of about $1.9\,mm$ (due to the higher supersaturation) in the work of [10], while in the present study the particles get only activated up to about $R = 1.4\,mm$.

Note, if the droplet formation would be completed within one cell in the flow direction of the domain, the final droplet size would be monodisperse. Hence, a polydisperse distribution can only be obtained (i) in case of droplet diffusion or (ii) if the nucleation process takes place over a larger number of cells in the flow direction. Since in our case diffusion was deactivated the latter case applies.
The nature of the obtained final droplet size distribution is comparable to a log-normal distribution according to Eq. (B.8) with $\mu' = 7 \cdot 10^{-6}$ and $\sigma = 0.625$. In this case the mean sizes, calculated with the moments according to Eq. (B.9), would be in the same range (i.e., $L_{10} = 8.5\,\mu m$, $L_{21} = 12.6\,\mu m$ and $L_{32} = 18.6\,\mu m$).

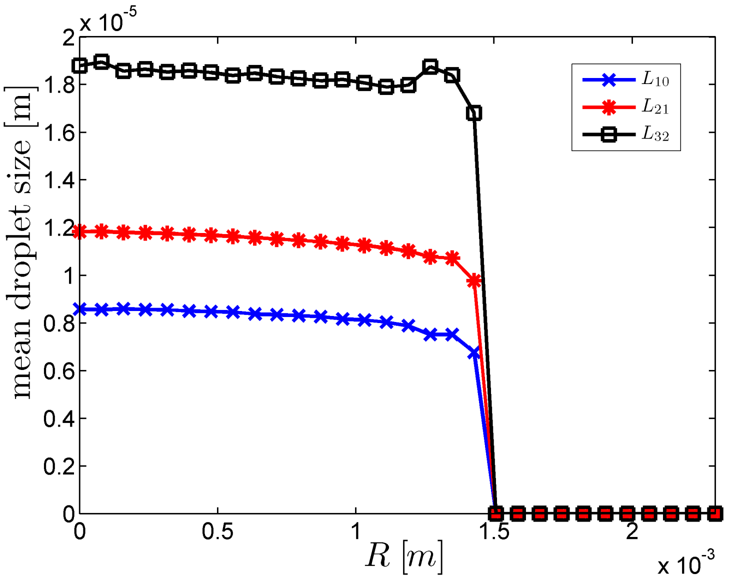


Figure 6.8: Final (mean) droplet size at the exit of the condenser without coupling: steady-state values (after $t = 1.2\,s$) of the mean droplet sizes L_{10}, L_{21} and L_{32} at $X = 0.243\,mm$. The seed particle size was set to $d_p = 40\,nm$. The simulation was run without coupling model and with a time step of $\Delta t = 10^{-6}\,s$.

6.5 Pipe Flow with Coupling

Finally we want run a simulation that includes the coupling model. Thus, all the implemented models, except diffusion, are employed in this case. The coupling model was activated by including the variable `'couplingModel'` in the *qmomProperties* dictionary of the pipe flow case. As in the previous section we ran the simulation with a time step of $\Delta t = 10^{-6}\, s$ up to an end time of $1.2\, s$.

The coupling model takes into account that the working fluid vapor is affected by the condensation process due to mass transport to the droplets and release of condensation heat. Therefore we would expect that the supersaturation is reduced compared to the uncoupled case in the previous section.
Fig. 6.9 shows the two-dimensional supersaturation profile in the condenser section. Also here, the highest supersaturation is located near the cylinder axis. However, compared with the uncoupled case (where we had the same supersaturation profile as without disperse phase, see Fig. 6.4) the profile seems more compressed in X-direction. This is because after the droplets have formed (which starts at an axial position of about $X \approx 0.195\, m$, see Fig. 6.5 (a)) they grow fast afterwards, and therefore, a lot of working fluid is condensed. As a consequence the supersaturation gets reduced.
The maximum value of the saturation ratio is 1.137 at $X = 0.2001\, m$ and $R = 0\, m$, which is essentially smaller than in the uncoupled case where the maximum value was 1.185 at $X = 0.2049\, m$. Thus, we observe a reduction of the maximum supersaturation of about $4\,\%$. Compared to the worst case scenario in [10, Fig. 16], where a maximum reduction of about $3\,\%$ in case of $40\, nm$ seed particles is documented, here we observe a higher reduction in S. This might be due to the different final particle sizes of $14\, \mu m$ in our case (Sauter mean diameter in case coupling was activated, see Fig. 6.10) and about $8\, \mu m$ in the calculations of Mamakos [10]. Note, in [10] the depletion in S was calculated using the mass of the droplets that have grown to their final droplet sizes (i.e. worst case scenario without considering coupling effects).

Due to the lower supersaturation, of course, also the condensation is reduced. Hence, we would expect that the droplets' growth is degraded compared to the uncoupled case. Fig. 6.10 displays the three mean droplet sizes L_{10}, L_{21} and L_{32}. The Sauter mean diameter L_{32} (which is the most important one) is reduced by $5\, \mu m$ from $19\, \mu m$ to about $14\, \mu m$ compared to the uncoupled case (see Fig. 6.8). L_{21} is lowered from 12 to about $9\, \mu m$, and the arithmetic mean size L_{10} has decreased from 8.5 to about $6.5\, \mu m$. Thus, we observe a general reduction of the final (mean) droplet size of approximately $20\,\%$.

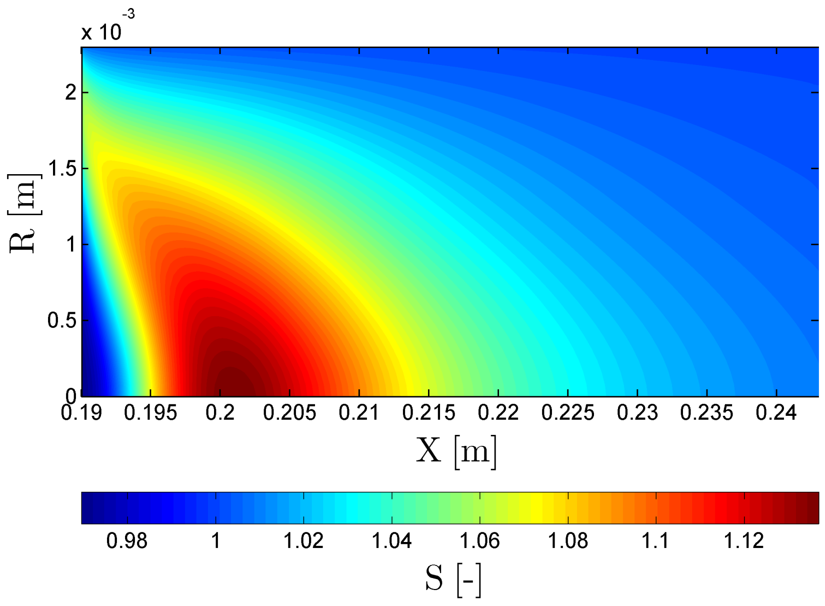


Figure 6.9: Two-dimensional supersaturation profile in the condenser at $t = 1.2\,s$ with activated coupling model.

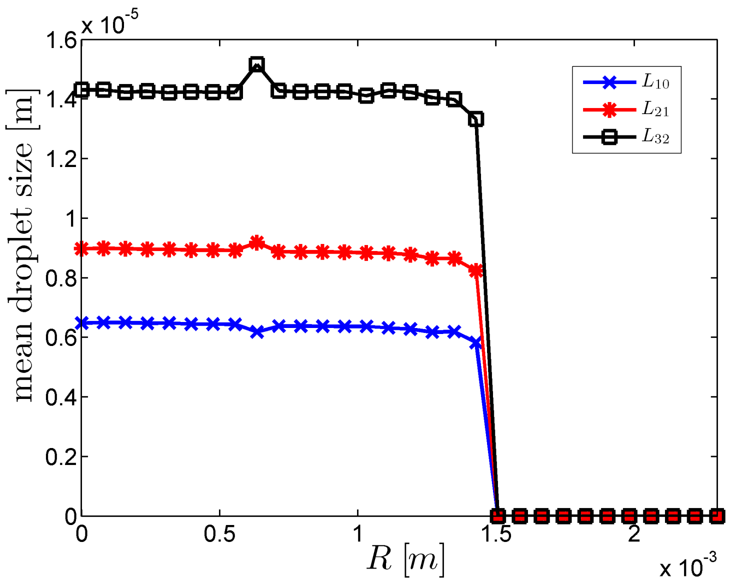


Figure 6.10: Final (mean) droplet size at the end of the condenser with coupling: The figure shows the steady-state values (at $t = 1.2\,s$) of the mean droplet size L_{10}, L_{21} and L_{32} at $X = 0.243\,mm$. Seed particle size $d_p = 40\,nm$, and $\Delta t = 10^{-6}\,s$.

7 Conclusions

This thesis aimed on the modeling of the condensation process inside the evaporator-condenser system of a CPC. In the evaporator the working fluid is vaporized and gets supersaturated in the condenser. The sub-micron particles, that should be counted, are sucked through the condenser where they act as seeds for heterogeneous nucleation. Once droplets have been formed on the seeds, vapor condenses and the droplets grow to a size in the range of a few microns, and thus, can be detected by light scattering. The purpose of the thesis was to implement a moment method in the open-source software tool 'OpenFOAM' in order to predict the time evolution of the droplet size distribution.

The key outcome of the thesis can be summarized as follows:

- Choice of the Method
  The simulation of the condensation process can be done by solving a population balance equation (PBE) for the disperse phase, i.e., the droplets. Since a direct solution of the PBE, or a lagrangian approach is very time consuming, a quadrature method of moments (QMOM) was chosen to model the evolution of a univariate PBE. The governing equations for the QMOM have been extracted from the book of Marchisio and Fox [8], and were summarized in see Sec. 3.

- Choice of the Physical Models
  Since the condensation process is divided into the formation and growth of droplets, for its description two physical models were required.

  - Nucleation Model
    For the formation of the disperse phase (i.e., the droplets), a heterogeneous nucleation model was used. This model assumes that the critical energy barrier for the formation of a stable droplet is reduced due to the presence of seed particles (see Eq. A.1). The interaction between the seed particle and the working fluid is lumped into a single parameter (i.e., the contact angle θ), which offers a pragmatic approach to reflect specific of different working fluid-particle systems.
  - Growth Model
    For the growth process the model of Abramzon and Sirignano [18] was used which requires a closure method for the system of moment transport equations. This is due to the size dependence of the growth rate. The underlying equations of the model were described in Sec. 3.5.1.
  - Diffusion Model
    The Stokes-Einstein diffusion model was presented in Sec. 3.5.2, and is now

implemented in the solver. This model can be used to describe the droplets' random motion, and thus, provides a correction of the dusty-gas assumption (i.e., no slip between fluid and particles). To model Stokes-Einstein diffusion behavior, a closure for the moment transport equations is needed. However, this model is relevant only for small droplets, i.e., in the early stages of droplet formation, since the diffusion rate decrases with increasing droplet diameter.

  – Coupling Model
    Based on the physical models for nucleation and growth, the governing equations for the mass and heat transfer between the continuous and the disperse phase were derived in Sec. 3.6. These equations formed the basis for the coupling model that is used to model the impact of the condensation process on (i) the concentration of the vaporized working fluid, and the (ii) gas temperature.

- Implementation in OpenFOAM - *qmomCloud*
  Based on the theoretical descriptions of Sec. 3 a new libary, called *qmomCloud*, was implemented and coupled to the existing *cpcFoamCompressible* solver. The structure of the library can be seen in Sec. 4.2. The library is implemented in a modular way, such that it can be easily extended, e.g., to model coalescence or breakage of droplets or particle agglomerates.

- The 3x3-Test Case
  To test the correct implementation of the *qmomCloud*, as well as the proper functionality of the nucleation and growth models, a simple 3x3-test case was constructed. Following the investigations of the TSI 3790 CPC, for all the tests within this thesis n-Butanol was used as working fluid. The number of moments to be tracked was limited to six due to the constraints in the standard eigenvalue computation in OpenFOAM. Nevertheless, as stated in [22], the use of three nodes and weights offers a sufficiently high accuracy.
  Within the 3x3-test case the two physical models were first investigated separately with different initial size distributions (monodisperse, log-normal, uniform, etc.). Afterwards they were tested in combination for the conditions that occur in a CPC, which means that a zero initial moment set becomes finite due to nucleation and is further modified due to growth. Finally the impact on the concentration and temperature of the working fluid vapor according to the coupling model was examined.

In case of pure growth of monodisperse droplets we found that the PD algorithm from Marchisio and Fox [8] has to be adapted in order to avoid singularities. Beside

this modification, a variety of changes and checks were implemented in order to avoid finite or complex nodes and weights. As a consequence, stable and reasonable results in case of pure growth could be obtained (see Fig. 5.3). Furthermore, it was found that the McGraw correction algorithm affects the growth behavior of monodisperse droplets in a way that the moment set gets distorted, and was therefore de-activated in further applications.
When both, the nucleation and the growth model, were activated it turned out that the nucleation is completed after a very short period (i.e., within one time step). Afterwards the moments (except m_0) change only due to growth, whereby the growth rate is higher for smaller initial droplet sizes (see Fig. 5.13).
Finally, the correct behavior of the coupling model was checked. It was found that the depletion of the working fluid vapor (i.e. the continuous phase) causes a decrease in the saturation ratio, while the temperature is increased due to the release of condensation heat. The heat and mass transfer between the vapor and the droplets (i.e., the disperse phase) stops if the supersaturation is reduced to one, as shown in Fig. 5.17.

- Pipe Flow Case
  Finally, the new solver was applied to the pipe flow case, i.e., a model of a CPC. The pipe flow case was first studied without disperse phase, then without coupling (but with dispersed phase), and finally all relevant physical models and the coupling were employed.

  – Results without Disperse Phase
    When deactivating the *qmomCloud*, the computation of disperse phase properties is deactivated and the solution for the continuous phase is given by the saturation ratio profile Fig. 6.4. With the calculated temperature profile, the Kelvin diameter can be calculated that indicates the minimum seed particle size for inducing the formation of droplets. It was found that the Kelvin diameter (Fig. 6.5) was below $40\,nm$ up to a radial position of approximately $R = 1.4\,mm$, i.e., primary particles of this size will be activated in this region. The maximum saturation ratio of 1.1849 was found at $X = 0.2049\,m$, while Mamakos [10] found a maximum value of 1.2177 at a position of $X = 0.2042\,m$ using the Graetz solution [9]. This difference is due to the fact that the solution of the Graetz problem only considers the condenser section with temperature boundary conditions at the inlet and at the wall of the pipe. Contrary to that, in our case we have also simulated the evaporation of the working fluid, and have therefore considered more realistic conditions at the condenser inlet.

  – Results with Disperse Phase - No Coupling
    When activating the *qmomCloud*, particle activation was found up to a radial

position of about $R = 1.4\,mm$ for a seed particle size of $40\,nm$. At the outlet of the condenser the droplets have grown to a final size (i.e., Sauter mean diameter) of about $19\,\mu m$. The arithmetic mean droplet diameter was found to be about $8\,\mu m$ which is in agreement with the final droplet size reported by Mamakos [10, Fig. 14].

– Results with Disperse Phase - Coupling
When employing the coupling model the saturation ratio in the condenser was significantly reduced. The maximum supersaturation was lowered by $3\,\%$ to 1.137 and was located at $X = 0.2001\,m$. As a consequence the final (mean) droplet size L_{32} was reduced to about $14\,\mu m$.

Generally, with the *qmomCloud* library a powerful tool for the modeling of condensation in the evaporator-condenser system of a CPC was generated. This tool can easily be extended in order to model additional effects, like coagulation and breakage.
However, the results for the pipe flow case were obtained by running the simulation with a relatively small time step of $10^{-6}\,s$. This was to avoid instabilities during the reconstruction procedure, which occurred due to the fine mesh (the domain of the pipe flow case was divided into 24000 cells) despite the usage of a stable numerical scheme for the moments. Hence, the simulations were quite time consuming (one simulation was run for up to 2 days in order to achieve steady state solutions).
In order to reduce the run time, a number of strategies could be followed in future work. For instance, the computational domain could be divided into a mesh with less cells, which would allow to run simulations with a larger time step (according Eq. (3.41) without diffusion). Furthermore, numerical schemes could be adopted for the nodes and weights instead of the moments, as suggested in [8, Sec. 8.3].
Note, in the course of this thesis no physical-space diffusion was taken into account for the pipe flow case. This was done because diffusional particle losses have insignificant effect in numerical calculations for the considered application, as stated in [10, p. 18]. Hence, we did not apply the Stokes-Einstein diffusion model within this thesis. Nevertheless, to overcome the instabilities in the moments for small time steps, a constant diffusion coefficient could be used (a non-zero diffusion term might stabilize the moment transport equations). Finally we suggest to investigate the pipe flow case with only two instead of four moments (i.e., only one node) which should lower the run time significantly (i.e., comparable to the monodisperse growth calculation Eq. (5.6)). When tracking only two moments, however, only the arithmetic mean droplet size can be calculated.

References

[1] "Commission Regulation (EC) No 692/2008 of 18 July 2008 implementing and amending Regulation To, No 715/2007 of the European Parliament and of the Council on type-approval of motor vehicles with respect Repair, emissions from light passenger and comm," tech. rep., 2008.

[2] B. Giechaskiel, M. Maricq, L. Ntziachristos, C. Dardiotis, X. Wang, H. Axmann, A. Bergmann, and W. Schindler, "Review of motor vehicle particulate matter emissions sampling and measurement : From smoke and filter mass to particle number," *J. Aerosol Sci.*, vol. 67, pp. 48–86, 2013.

[3] J. Andersson and B. Giechaskiel, "Particle Measurement Programme (PMP) light-duty inter-laboratory correlation exercise (ILCE-LD) final report," *Inst. Environ. ...*, 2007.

[4] J. Andersson, A. Mamakos, B. Giechaskiel, M. Carriero, and G. Martini, "Particle Measurement Programme (PMP) Heavy-duty Inter-laboratory Correlation Exercise (ILCE _ HD) Final Report," tech. rep., 2010.

[5] B. Giechaskiel and A. Bergmann, "Validation of 14 used, re-calibrated and new TSI 3790 condensation particle counters according to the UN-ECE Regulation 83," *J. Aerosol Sci.*, pp. 1–28, 2011.

[6] D. L. Marchisio, A. Barresi, B. G., and R. Fox, "Comparison between the Classes Method and the Quadrature Method of Moments for Multiphase Systems," in *8th Conf. "Multiphase flow Ind. plants", Alba (Italy), Sept. 18-20*, p. 16, 2002.

[7] D. L. Marchisio, J. T. Pikturna, R. O. Fox, R. D. Vigil, D. Scienza, P. Torino, and C. Duca, "Quadrature Method of Moments for Population-Balance Equations," vol. 49, no. 5, pp. 322–334, 2003.

[8] D. L. Marchisio and R. O. Fox, *Computational Models for Polydisperse Particulate and Multiphase Systems.* 2013.

[9] C. Housiadas, F. E. Larrode, and Y. Drossinos, "Numerical evaluation of the Graetz series," *Int. J. Heat Mass Transf.*, pp. 2902–2906.

[10] A. Mamakos, B. Giechaskiel, Y. Drossinos, D. Lesueur, G. Martini, and A. Krasenbrink, *Calibration and Modeling of PMP compliant Condensation Particle Counters.* 2011.

[11] B. Giechaskiel, X. Wang, D. Gilliland, and Y. Drossinos, "The effect of particle chemical composition on the activation probability in n-butanol condensation particle counters," *J. Aerosol Sci.*, vol. 42, no. 1, pp. 20–37, 2011.

[12] A. Mamakos, B. Giechaskiel, and Y. Drossinos, “Experimental and Theoretical Investigations of the Effect of the Calibration Aerosol Material on the Counting Efficiencies of TSI 3790 Condensation Particle Counters,” *Aerosol Sci. Technol.*, vol. 47, pp. 11–21, Jan. 2013.

[13] N. Fuchs and R. Bradley, *Evaporation and Droplet Growth in Gaseous Media.* 1959.

[14] K.-H. Ahn and B. Y. Liu, “Particle activation and droplet growth processes in condensation nucleus counterI. Theoretical background,” *J. Aerosol Sci.*, vol. 21, pp. 249–261, Jan. 1990.

[15] P. Taylor, D. A. Terry, R. Mcgraw, and R. H. Rangel, “Aerosol Science and Technology Method of Moments Solutions for a Laminar Flow Aerosol Reactor Model Method of Moments Solutions for a Laminar Flow,” no. March 2014, pp. 37–41, 2010.

[16] D. L. Marchisio and R. O. Fox, “Solution of population balance equations using the direct quadrature method of moments,” *J. Aerosol Sci.*, vol. 36, pp. 43–73, Jan. 2005.

[17] “OpenFOAM - The Open Source Computational Fluid Dynamics (CFD) Toolbox.” `http://www.openfoam.org/`. [accessed 2014-09-27].

[18] B. Abramzon and W. Sirignano, “Droplet vaporization model for spray combustion calculations,” *Int. J. Heat Mass Transf.*, vol. 32, pp. 1605–1618, Sept. 1989.

[19] A. Gerber and A. Mousavi, “Application of quadrature method of moments to the polydispersed droplet spectrum in transonic steam flows with primary and secondary nucleation,” *Appl. Math. Model.*, vol. 31, pp. 1518–1533, Aug. 2007.

[20] W. C. Hinds, *Aerosol Technology: Properties, Behavior, and Measurement of Airborne Particles.* 2 ed., 1999.

[21] E. Paterson, “Applications and case setup.” `http://web.student.chalmers.se/groups/ofw5/Basic_Training/AppsAndCases.pdf`. [Online; accessed 2014-09-14].

[22] D. L. Marchisio and R. O. Fox, *Mulitphase Reacting Flows: Modelling and Simulation.* 2007.

[23] S. Perry, R. H. Perry, D. W. Green, and J. O. Maloney, *Perry’s Chemical Engineer’s Handbook Seventh Edition.* 1997.

[24] VDI, *VDI-Wärmeatlas.* (GVC), Verein Deutscher Ingenieure VDI-Gesellschaft Verfahrenstechnik und Chemieingenieurwesen, 10 ed., 2006.

[25] J. A. Dean, *Lange’s Handbook of Chemistry.* 1944.

A Appendix

A Heterogeneous Nucleation: Impact of f_g

It is intuitive that nucleation of droplets is facilitated when seed particles act as condensation nuclei. This is referred to as heterogeneous nucleation, which is explained in Sec. 3.5.4. Due to the presence of seed particles the energy barrier for droplet formation is reduced by a factor f_g

$$\Delta G^*_{het} = f_g \cdot \Delta G^*_{hom} \qquad 0 \leq f_g \leq 1 \tag{A.1}$$

f_g takes into account the contact angle θ between a critical cluster of vapour molecules and the curved particle surface. The contact angle characterizes the affinity of the seed particles for the droplets and takes values between 0 and 180°.
The higher the affinity of the particle for the working fluid, the lower the contact angle and f_g. Thus, the lower is the energy barrier for particle activation. This behavior is illustrated in Fig. A.1.
The multiplicative factor f_g can be obtained by geometrical considerations as [10]

$$f_g = \frac{1}{2}\left\{1 + \left(\frac{1 - X'\cos\theta}{g}\right)^3 + X'^3\left[2 - 3\left(\frac{X' - \cos\theta}{g}\right) + \left(\frac{X' - \cos\theta}{g}\right)^3\right] + 3X'^2\cos\theta\left(\frac{X' - \cos\theta}{g}\right)^3\right\} \tag{A.2}$$

with $X' = \frac{d_p}{2r^*}$ and $g = \sqrt{1 + X'^2 - 2X'\cos\theta}$.

In the case of perfect wetting the contact angle is zero. For seed particles bigger than the critical cluster ($d_p \geq 2r^*$) also f_g is equal to zero and the energy barrier for particle activation vanishes. For seed particles smaller than the critical cluster $f_g > 0$. For perfect wetting ($\theta = 0°$) f_g is given by

$$f_g = \begin{cases} 0 & \text{for } d_p \geq 2r^* \\ (1 - X')^2\,(1 + X') & \text{for } d_p \leq 2r^* \end{cases} \tag{A.3}$$

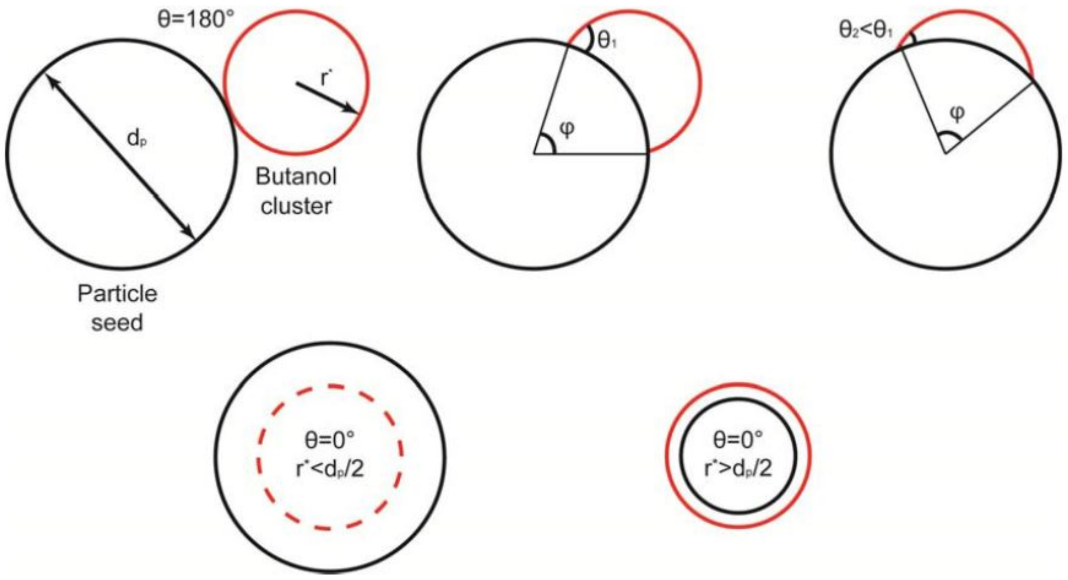


Figure A.1: Illustration of the effect of the contact angle on the heterogeneous nucleation mechanism: nucleation behavior of a butanol droplet on a seed particle at different contact angles. Figure taken from [10].

We now investigate the variation of nucleation rate with f_g. In order to do so, we assume J^0_{het} in Eq. (3.54) to be one, thus, the nucleation rate is given by

$$J_{het} = \exp\left(-\frac{f_g \cdot \Delta G^*_{hom}}{k_B T_f}\right) \tag{A.4}$$

As it can be seen in Fig. A.2, the nucleation rate quickly drops towards zero with increasing values of f_g. For $f_g = 0$ the nucleation rate is given by J^0_{het}. Note that for the very small value of $f_g \approx 5 \cdot 10^{-4}$, J_{het} is already reduced to $J^0_{het}/2$. Thus the values of f_g should be very close to zero to enhance droplet formation using seed particles.

The main influence parameters to f_g, and thus, to the nucleation rate J, are the contact angle and the ratio of the critical cluster size and the seed particle diameter. As shown in Fig. A.3, for perfect wetting ($\theta = 0°$) the nucleation rate quickly increases when the seed particle is bigger than the critical droplet size. For higher contact angles J rises more slowly, and for contact angles higher than 120°, J even remains close to zero.
The figure shows that on the one hand it is crucial to have large enough seed particles to ensure that nucleation on the particles takes place. On the other hand the affinity of the seed particles for the droplets, that determines the contact angle, strongly affects the nucleation rate.

In general one can conclude that the higher the affinity of the seed particles for the droplets and the larger the seed particles, the easier will be the formation of droplets.

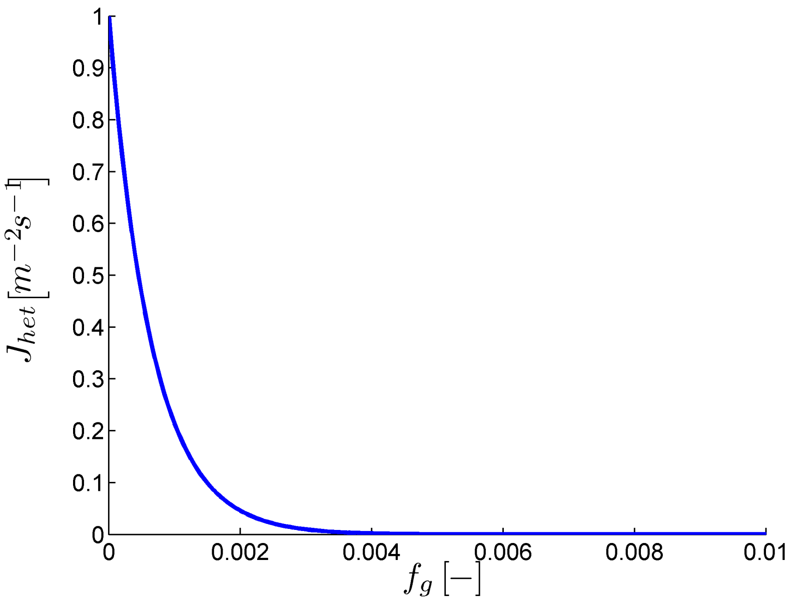


Figure A.2: Heterogeneous nucleation rate as a function of the multiplicative factor f_g that can take values between 0 and 1. J^0_{het} was set to $1\,m^{-2}s^{-1}$.

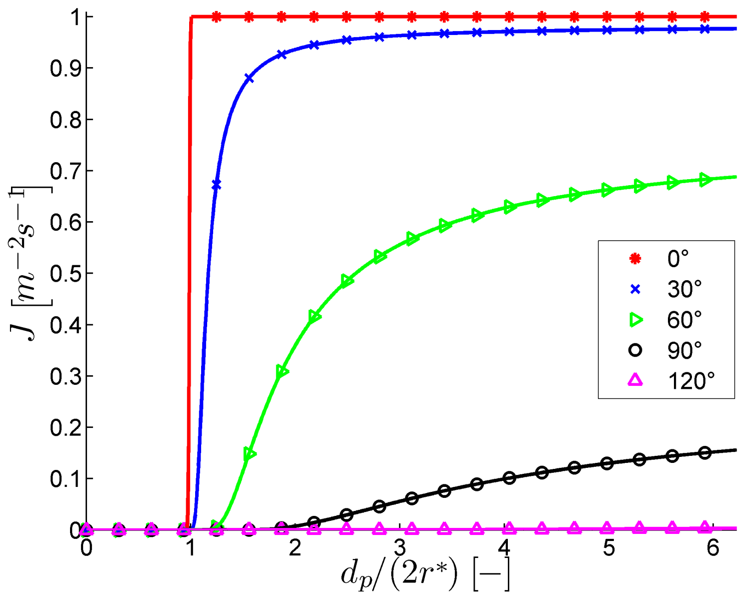


Figure A.3: Nucleation rate J as a function of the ratio $d_p/(2r^*)$ for different contact angles according to equations Eq. (A.2) and Eq. (A.4).

B Additional Equations

- Navier-Stokes Equation
  This equation describes fluid motion in the context of CFD.

$$\frac{\partial}{\partial t}(\rho \boldsymbol{u}) + \nabla \cdot (\rho \boldsymbol{u}\boldsymbol{u}) = \nabla \cdot \left[\mu\left(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T\right)\right] - \nabla p + \rho g \tag{B.5}$$

  For compressible flows the full set of conservation equations consist of mass, momentum, and (thermal and kinetic) energy conservation, as well as the equation of state.

- Energy Conservation
  The energy conservation is written as a transport equation for the total enthalpy, i.e., thermal and kinetic energy. Since the flow velocity is small, the kinetic energy can be neglected, and we get

$$\frac{\partial}{\partial t}(\rho h) + \nabla \cdot (\rho \boldsymbol{u} h) - \frac{\partial}{\partial t} p = \nabla \cdot \left(\frac{\lambda}{c_p}\nabla h\right) + \dot{\mathcal{S}}_{heat} \tag{B.6}$$

  Note, that enthalpy transport due to diffusion is neglected, since the vapor mass fraction is small.

- Microscale description of a polydisperse multiphase system [8]:

$$\begin{aligned}\frac{\partial n}{\partial t} + \frac{\partial}{\partial \boldsymbol{x}}\left[\boldsymbol{v_p} n + (\boldsymbol{v_p} - \boldsymbol{v_f})\frac{\partial(\xi_{f1} n)}{\xi_{f1}}\right] = -\frac{\partial}{\partial \boldsymbol{v_p}}\left[\left(\langle \boldsymbol{A}_{fp}\rangle_1 + \langle \boldsymbol{A}_p\rangle_1\right) n\right] \\ -\frac{\partial}{\partial \boldsymbol{\xi_p}}\left(\langle \boldsymbol{G}_p\rangle_1 n\right) \\ -\frac{\partial}{\partial \boldsymbol{v_f}}\left[\left(\langle \boldsymbol{A}_{pf}\rangle_1 + \langle \boldsymbol{A}_f\rangle_1\right) n\right] - \frac{\partial}{\partial \boldsymbol{\xi_f}}\left(\langle \boldsymbol{G}_p\rangle_1 n\right) + \mathcal{S}_1 \\ -\frac{\partial}{\partial \boldsymbol{v}_f}\left[\frac{1}{\tau_{pf}}\left(1 - \frac{\rho_v \xi_{p1}}{\rho_p \xi_{f1}}\right)\boldsymbol{v}_f n\right] \\ -\frac{\partial}{\partial \xi_{f1}}\left[\frac{1}{\tau_{pf}}\left(\frac{\rho_v}{\rho_p}\xi_{p1} - \xi_{f1}\right) n\right]\end{aligned} \tag{B.7}$$

- Log-normal distribution:
  The normalized NDF in case of a log-normal distribution is

$$n(\xi) = \frac{1}{\sqrt{2\pi}\xi\sigma'}\exp\left(-\frac{(\ln(\xi/\mu'))^2}{2\sigma'^2}\right) \qquad \text{for } x > 0 \tag{B.8}$$

The moments of the log-normal distribution can be calculated via

$$m_k = \exp\left(k\ln(\mu') + \frac{k^2\sigma'^2}{2}\right) \tag{B.9}$$

C Species Parameters

For all the simulations within this thesis n-butanol was used as working fluid, which are summarized in the following.

- molecular weight from [10]: $MW_v = 74.123 \cdot 10^{-3}\, \frac{kg}{mol}$
- critical temperature from [23, p. 2-137]: $T_c = 563.05\, K$
- liquid density from [23, p. 2-95]: (T in K)

$$\rho_l(T) = 71.5287 \cdot \left(0.266^{\left(1+\left(1-\frac{T}{563.05}\right)^{0.24419}\right)}\right)^{-1}\; kg/m^3 \tag{C.10}$$

- vapor molecular diffusion coefficient from [10]: (T in K, p in N/m^2)

$$D_v(p,T) = 4.26199 \cdot 10^{-5} \cdot T^{1.75} \cdot p^{-1}\, m^2/s \tag{C.11}$$

- vapor pressure from [23, p. 2-51]: (T in K)

$$p_v(T) = \exp\left[93.173 - \frac{9185.9}{T} - 9.7464\ln(T) + 4.7796 \cdot 10^{18} T^6\right] \tag{C.12}$$

- surface tension [24, p. Dca 45]: (T and T_c in K)

$$\sigma(T) = 0.04839\left(1 - \frac{T}{T_c}\right)^{0.91063}\; kg/s^2 \tag{C.13}$$

  Note, in Mamakos [10] the surface tension according to [25, p. 5.91] was used: (T in K)

$$\sigma(T) = \frac{27.18 - 0.0898(T - 273.15)}{1000}\, kg/s^2 \tag{C.14}$$

- enthalpy of vaporization from [23, p. 2-157]: (T and T_c in K)

$$h_v(T) = 6.7390 \cdot 10^7\left(1 - \frac{T}{T_c}\right)^{0.173+0.2915\frac{T}{T_c}}\; J/kmol \tag{C.15}$$

D Definitions

- disperse phase: finely dispersed droplets (=continous phase), e.g. colloids
- phase space: is the space of all possible particle states. In the mesoscale description, the phase space ist defined by the internal coordinates, thus the infinitesimal phase space volume $d\xi$
- seed particles (also called primary particles): particles that act as nucleation seed with the consequence that the energy barrier for the formation of a droplet is decreased
- dusty-gas model: in this model it is assumed that the particles' velocity field is equal to that of the continuous phase. This assumption is valid in case of a small Stokes number.

E Code Files

E.1 Main Files of the *qmomCloud* Library

File: /home/mario/OpenFOAM/mario-2..../cpcFoam/qmomCloud/qmomCloud.C Page 1 of 6

```
/*---------------------------------------------------------------------------*\
License

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

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

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

    Copyright (C) 2014- Stefan Radl, TU Graz, Austria

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

#include "qmomCloud.H"
//TODO: include declarations of subModels
#include "qmomSolver.H"
#include "transportModel.H"
#include "physicalModel.H"
#include "error.H"
//#include "couplingModel.H"

// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * //
Foam::qmomCloud::qmomCloud
(
    const fvMesh& mesh,
    const double  Csat,
    const double  Ysat
)
:
    mesh_(mesh),
    qmomProperties_
    (
        IOobject
        (
            "qmomProperties",
            mesh_.time().constant(),
            mesh_,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        )
    ),
    verbose_(false),
    ignore_(false),
    liqDictionary_
    (
        IOobject
        (
            "liqProperties",
            mesh.time().constant(),
            mesh,
            IOobject::MUST_READ_IF_MODIFIED,
            IOobject::NO_WRITE
        )
    ),
    liquids_
    (
        liqDictionary_.subDict("species")
    ),
    xLiquid_
    (
        "molFractions",
        liqDictionary_.subDict("composition"),
        liquids_.size()
    ),
    thermoDictionary_
    (
        IOobject
        (
            "thermophysicalProperties",
            mesh_.time().constant(),
            mesh_,
            IOobject::MUST_READ_IF_MODIFIED,
            IOobject::NO_WRITE
        )
    ),
    pRef_
    (
         thermoDictionary_.lookup("pRefSpecies")
    ),
    CSat_(Csat),
    YSat_(Ysat),
    transportModelList_(qmomProperties_.lookup("transportModels")),
    physicalModelList_(qmomProperties_.lookup("physicalModels")),
    couplingModel_(NULL),
    qmomSolver_(NULL),
    momSource_
    (
        IOobject
        (
            "momSource",
            mesh_.time().timeName(),
            mesh_,
            IOobject::READ_IF_PRESENT,
            IOobject::NO_WRITE
        ),
        mesh_,
        dimensionedScalar("zero", dimensionSet(0,0,-1,0,0), scalar(0.0)) //1/s
    ),
    massSource_
    (
        IOobject
        (
            "massSource",
            mesh_.time().timeName(),
            mesh_,
            IOobject::READ_IF_PRESENT,
            IOobject::NO_WRITE
        ),
        mesh_,
        dimensionedScalar("zero", dimensionSet(1,-3,-1,0,0), scalar(0.0)) //kg/m³/s
    ),
    enthalpySource_
    (
        IOobject
        (
            "enthalpySource",
            mesh_.time().timeName(),
            mesh_,
            IOobject::READ_IF_PRESENT,
            IOobject::NO_WRITE
        ),
        mesh_,
        dimensionedScalar("zero", dimensionSet(1,-1,-3,0,0), scalar(0.0)) //W=kgm^2/s^3
    ),
    velFieldName_(qmomProperties_.lookup("velFieldName")),
    U_(mesh_.lookupObject<volVectorField> (velFieldName_)),
    phi_
```

```
    (
       (
           linearInterpolate(U_) & mesh_.Sf()
       )
    ),
    requiresPDF_(false),
    doCoupling_(false)

{
    #include "versionInfo.H"

    if (qmomProperties_.found("verbose")) verbose_=true;
    if (qmomProperties_.found("ignore")) ignore_=true;

    transportModels_ = new autoPtr<transportModel>[nrtransportModels()];
    for (int i=0;i<nrtransportModels();i++)
    {
        transportModels_[i] = transportModel::New
        (
            qmomProperties_,
            *this,
            transportModelList_[i],
            i
        );
    }

    physicalModels_ = new autoPtr<physicalModel>[nrPhysicalModels()];
    for (int i=0;i<nrPhysicalModels();i++)
    {
        physicalModels_[i] = physicalModel::New
        (
            qmomProperties_,
            *this,
            physicalModelList_[i],
            i
        );

        if( physicalModels_[i]->requiresPDF() )
            requiresPDF_ = true;
    }

    if (qmomProperties_.found("couplingModel"))
    {
        doCoupling_=true;
        Info << "qmomCloud couplingModel: Will perform back-coupling to fluid equations." << endl;
    }

    qmomSolver_ =
        qmomSolver::New
        (
            qmomProperties_,
            *this
        );

    #include "endVersionInfo.H"

}

// * * * * * * * * * * * * * * * * Destructors  * * * * * * * * * * * * * * //
Foam::qmomCloud::~qmomCloud()
{
}
// * * * * * * * * * * * * * * * private Member Functions  * * * * * * * * * * * * * //


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

```

```
//void Foam::qmomCloud::setVectorAverages()
//{
//    if(verbose_) Info << "- setVectorAverage(Us,velocities_,weights_)" << endl;
//    averagingM().setVectorAverage
//    (
//        averagingM().UsNext(),
//        velocities_,
//        particleWeights_,
//        averagingM().UsWeightField(),
//        NULL //mask
//    );
//    if(verbose_) Info << "setVectorAverage done." << endl;
//}

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


// * * * * * * * * * * * * * * * ACCESS  Functions  * * * * * * * * * * * * //

const transportModel& Foam::qmomCloud::transportM(int i)
{
    return transportModels_[i];
}

const physicalModel& Foam::qmomCloud::physicalM(int i)
{
    return physicalModels_[i];
}

int Foam::qmomCloud::nrtransportModels()
{
    return transportModelList_.size();
}

int Foam::qmomCloud::nrPhysicalModels()
{
    return physicalModelList_.size();
}

//****************************************
//Prepare moment sources
void Foam::qmomCloud::computeMomSource(int kthMom_)
{
    //Set to zero
    momSource_ *= 0.0;

    //Update physical models and sum up
    //source for kthMoment
    for(int iModel=0; iModel < nrPhysicalModels(); iModel++)
    {
        //Prepare the source terms before computing source for 0-th moment
        if(kthMom_==0)
            physicalM(iModel).update();

        //Sum-up the source terms
        momSource_ += physicalM(iModel).source(transportModels_, kthMom_);
    }
    return;

}

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

//Prepare mass sources
void Foam::qmomCloud::computeMassSource()
{
    //Set to zero
```

File: /home/mario/OpenFOAM/mario-2..../cpcFoam/qmomCloud/qmomCloud.C Page 5 of 6

```
    massSource_ *= 0.0;

    //Check if physical model is up to date
    for(int iModel=0; iModel < nrPhysicalModels(); iModel++)
    {
        if(physicalM(iModel).isUpToDate())
            massSource_ -= physicalM(iModel).massSource(transportModels_);
        else
            FatalError << "Physical model with id " << iModel << " is not up to date. This is bad."
<< abort(FatalError) ;
    }
    return;

}

void Foam::qmomCloud::computeEnthalpySource()
{
    //Set to zero
    enthalpySource_ *= 0.0;

    //Check if physical model is up to date
    for(int iModel=0; iModel < nrPhysicalModels(); iModel++)
    {
        if(physicalM(iModel).isUpToDate())
            enthalpySource_ += physicalM(iModel).energySource(transportModels_);
        else
            FatalError << "Physical model with id " << iModel << " is not up to date. This is bad."
<< abort(FatalError) ;
    }
    return;

}

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

// * * *   Evolve    * * * //
bool Foam::qmomCloud::evolve
(
//    volScalarField& alpha,
//    volVectorField& Us,
//    volVectorField& U
)
{

    bool doCouple = false;
    if(!ignore())
    {

        // A - reconstruct distribution by QMOM Solver if necessary
        qmomS().solve(transportModels_);

        // B1 - Update convective flux
        phi_ = linearInterpolate(U_) & mesh_.Sf();

        // B2 - Update Transport models
        for(int iTModel=0; iTModel < nrtransportModels(); iTModel++)
        {
                transportM(iTModel).update(phi_);
                transportM(iTModel).bound( transportModels_ );
        }

        //Update the sources for the coupling
        computeMassSource();
        computeEnthalpySource();

    }//end ignore
    return doCouple;
```

```
}

//////////////////////////////////////////////////////////////////////////////

void  Foam::qmomCloud::resetStatus()
{
        for(int iModel=0; iModel < nrPhysicalModels(); iModel++)
        {
            physicalM(iModel).resetStatus();
        }
}

// * * * * * * * * * * * * * * * * *  IOStream operators * * * * * * * * * * * //

#include "qmomCloudIO.C"

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

File: /home/mario/OpenFOAM/mario-2..../cpcFoam/qmomCloud/qmomCloud.H Page 1 of 4

```
/*---------------------------------------------------------------------------*\
License

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

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

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

    Copyright (C) 2014- Stefan Radl, TU Graz, Austria

Application / Class
    qmomCloud

Description
    Core class for particle clouds to be described with a quadrature
    method of moments (QMOM)

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

#ifndef qmomCloud_H
#define qmomCloud_H

// choose version
//#include "OFversion.H" TODO: list compatible OF versions

#include "fvCFD.H"
#include "IFstream.H"
#include "liquidProperties.H"
#include "liquidMixtureProperties.H"

#if defined(version21) || defined(version16ext)
    #include "turbulenceModel.H"
#elif defined(version15)
    #include "RASModel.H"
#endif

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

namespace Foam
{

// forward declarations
class transportModel;
class qmomSolver;
class physicalModel;
class couplingModel;


/*---------------------------------------------------------------------------*\
                           Class qmomCloud Declaration
\*---------------------------------------------------------------------------*/

class qmomCloud
{

// protected data
protected:

    const fvMesh& mesh_;
```

File: /home/mario/OpenFOAM/mario-2..../cpcFoam/qmomCloud/qmomCloud.H Page 2 of 4

```

    const IOdictionary      qmomProperties_;

    bool verbose_;

    bool ignore_;

    //Liquid Properties
    const IOdictionary            liqDictionary_;
    const liquidMixtureProperties liquids_;
    const scalarField             xLiquid_;
    const IOdictionary            thermoDictionary_;
    const dimensionedScalar       pRef_;

    //Reference Gas Properties
    const double                  CSat_;
    const double                  YSat_;


    //List of submodels to be used
    const wordList transportModelList_;
    const wordList physicalModelList_;

    autoPtr<transportModel>*  transportModels_;
    autoPtr<physicalModel>*   physicalModels_;

    //the control to the coupling
    autoPtr<couplingModel>    couplingModel_;

    //the solver for reconstructing the particle PDF
    autoPtr<qmomSolver> qmomSolver_;

    //Moment Source Field
    mutable volScalarField  momSource_;

    //Exchange fields
    mutable volScalarField  massSource_;       //sum of all mass sources from couplingModel
    mutable volScalarField  enthalpySource_;   //sum of all enthalpySources from couplingModel

    word velFieldName_;
    const volVectorField& U_;
    mutable surfaceScalarField phi_;

    mutable bool requiresPDF_;

    mutable bool doCoupling_;

// Protected member functions
//    virtual void setForces();


public:

    friend class qmomSolver;

// Constructors

    //- Construct from mesh and a list of particles
    qmomCloud
    (
            const fvMesh& mesh,
            const double  Csat,
            const double  Ysat
    );

    //- Destructor
    virtual ~qmomCloud();
```

```

// public Member Functions

    // Access
       virtual const    transportModel& transportM(int);
       virtual const    physicalModel&  physicalM(int);

       virtual int      nrtransportModels();
       virtual int      nrPhysicalModels();

       inline const     qmomSolver& qmomS() const;

       inline bool      verbose() const;

       inline bool      doCoupling() const { return doCoupling_;};

       inline const bool& ignore() const;

       inline const IOdictionary& qmomProperties() const;
       inline const IOdictionary& liqDictionary() const;
       inline const liquidMixtureProperties& liquids() const;
       inline const scalarField& xLiquid() const;
       inline const dimensionedScalar& pRef() const;

       inline const double& CSat() const;
       inline const double& YSat() const;

       inline const               fvMesh& mesh() const;

       inline const wordList& transportModels();


    // Write

      // write qmomCloud internal data
        virtual bool evolve();

        bool requiresPDF() const {return requiresPDF_;}; //return true if a physical model requires
the full PDF

    // Sources for Moments
        void computeMomSource(int modelID_);
        const volScalarField& momSource(){return momSource_;};

    // Sources for Mass and energy
        void  computeMassSource();
        const volScalarField& massSource(){return massSource_;};

        void  computeEnthalpySource();
        const volScalarField& enthalpySource(){return enthalpySource_;};

        void  resetStatus();

    // functions
        //NONE AT THE MOMENT
};


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

} // End namespace Foam

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

#include "qmomCloudI.H"

#endif
```

File: /home/mario/OpenFOAM/mario-2..../cpcFoam/qmomCloud/qmomCloud.H Page 4 of 4

```

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

File: /home/mario/OpenFOAM/mario-2....entTransport/momentTransport.C Page 1 of 3

```
/*---------------------------------------------------------------------------*\
License

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

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

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

    Copyright (C) 2014- Stefan Radl, TU Graz, Austria

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

#include "error.H"
#include "momentTransport.H"
#include "addToRunTimeSelectionTable.H"
#include "qmomSolver.H"
#include <sstream>
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

namespace Foam
{

//Helper function
inline string intToString (int a)
{
    std::ostringstream temp;
    temp<<a;
    return temp.str();
}

// * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //
defineTypeNameAndDebug(momentTransport, 0);

addToRunTimeSelectionTable
(
    transportModel,
    momentTransport,
    dictionary
);

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

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

// Construct from components
momentTransport::momentTransport
(
    const dictionary& dict,
    qmomCloud& sm,
    int modelID
)
:
    transportModel(dict,sm,modelID),
    propsDict_(dict.subDict(typeName + "Props")),
    verbose_(false),
    fieldName_(propsDict_.lookup("fieldName")),
    m_
    (   IOobject
        (
            fieldName_+intToString(modelID),
            sm.mesh().time().timeName(),
            sm.mesh(),
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        sm.mesh()
    ),
    modelID_(modelID),
    Dp_0_
    (
        dimensionedScalar("Dp0",
                          dimensionSet(0, 2, -1, 0, 0),
                          readScalar(propsDict_.lookup("Dp0"))
                         )
    ),
    limitMomentAbove_(99),
    momentLimitFactor_(1.0)
{
    if (propsDict_.found("verbose")) verbose_=readBool(propsDict_.lookup("verbose"));

    if (propsDict_.found("limitMomentAbove"))
    {
        limitMomentAbove_ = readScalar(propsDict_.lookup("limitMomentAbove"));
        if(limitMomentAbove_<2)
            FatalError << "limitMomentAbove must be >=2 in order for the momentum limiter to work"
<< abort(FatalError) ;

        momentLimitFactor_= readScalar(propsDict_.lookup("momentLimitFactor"));
        if(momentLimitFactor_<=1.01)
            FatalError << "momentLimitFactor_must be >1.01 in order for the momentum limiter to
work" << abort(FatalError) ;
        Info << "...detected moment limiter. Will limit moments >= "
             << limitMomentAbove_
             << " with factor " << momentLimitFactor_ << endl;
    }
}


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

momentTransport::~momentTransport()
{}

// * * * * * * * * * * * * * * * * Member Fct  * * * * * * * * * * * * * * * //
void momentTransport::update(surfaceScalarField phi) const
{
    //Set the sources for the moments
    particleCloud_.computeMomSource(modelID_);

    //Assemble transport equation
    fvScalarMatrix MEqn
    (
        fvm::ddt(m_)
      + fvm::div(phi, m_,"div(phi,m)")    //phi is just velocity here
      ==
        fvm::laplacian(Dp_0_, m_,"laplacian(Dp,m)")
      + particleCloud_.momSource()
    );

    if(verbose_)
    {
        dimensionedScalar  totalSource = gSum(particleCloud_.momSource().internalField()*m_.mesh().V
());
        dimensionedScalar  totalVolume = gSum(m_.mesh().V());

        Info << "volume-average momSource[" << modelID_ << "]: " << (totalSource/totalVolume).value
```

```
() << endl;
    }

    //solve Eqn.
    MEqn.solve(m_.mesh().solver("m"));

    return;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
void momentTransport::bound(autoPtr<transportModel>* transportModels) const
{

    //lower bound
    m_= max(0.0,m_);

    //Upper Limiter
    if(modelID_ >= limitMomentAbove_)
    {
      Info << "...limiting moment with id " << modelID_ << endl;

      forAll(m_.internalField(),cellI)
      {
          //estimate current moment from previous two moments
          double m_0 = transportModels[0]->m().internalField()[cellI];
          double m_Minus1 = transportModels[modelID_-1]->m().internalField()[cellI];

          double m_expected = m_0
                            * powf(
                                      m_Minus1 / (m_0+SMALL),
                                      double(modelID_) / double(modelID_-1.0)
                                  );

          if(verbose_)
            Info << "m0: "          << m_0
                 << ", m[x-1]: "    << m_Minus1
                 << ", m_exp[x]: "  << m_expected
                 << ", m_curr[x]: " << m_.internalField()[cellI] << endl;

          if(  ( m_.internalField()[cellI] > m_expected * momentLimitFactor_ ) //too high
             ||( m_.internalField()[cellI] < m_expected / momentLimitFactor_ ) //too low
            )
                m_.internalField()[cellI] = m_expected; //reset to expected value in case value
above the limit


       }
    }


}

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

} // End namespace Foam

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

File: /home/mario/OpenFOAM/mario-2....lModel/nucleation/nucleation.C Page 1 of 5

```
/*---------------------------------------------------------------------------*\
License

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

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

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

    Copyright (C) 2014- Stefan Radl, TU Graz, Austria

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

#include "error.H"
#include "nucleation.H"
#include "transportModel.H"
#include "qmomSolver.H"
#include "addToRunTimeSelectionTable.H"
#include <sstream>
#include <math.h>
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

namespace Foam
{

// * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //
defineTypeNameAndDebug(nucleation, 0);

addToRunTimeSelectionTable
(
    physicalModel,
    nucleation,
    dictionary
);

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

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

// Construct from components
nucleation::nucleation
(
    const dictionary& dict,
    qmomCloud& sm,
    int modelID
)
:
    physicalModel(dict,sm,modelID),
    propsDict_(dict.subDict(typeName + "Props")),
    verbose_(false),
    SFieldName_(propsDict_.lookup("SName")),
    S_(sm.mesh().lookupObject<volScalarField> (SFieldName_)),
    rStar_
    (
        IOobject
        (
           "rStar",
            sm.mesh().time().timeName(),
            sm.mesh(),
            IOobject::READ_IF_PRESENT,
            IOobject::AUTO_WRITE
        ),
        sm.mesh(),
        dimensionedScalar("zero", dimensionSet(0,1,0,0,0), scalar(0.0))
    ),
    J_
    (
        IOobject
        (
           "JNuc",
            sm.mesh().time().timeName(),
            sm.mesh(),
            IOobject::READ_IF_PRESENT,
            IOobject::AUTO_WRITE
        ),
        sm.mesh(),
        dimensionedScalar("zero",
                          dimensionSet(0, 0, -1, 0, 0),
                          scalar(0.0)
                         )
    ),
    J_0_
    (
        dimensionedScalar("J0",
                          dimensionSet(0, 0, -1, 0, 0),
                          readScalar(propsDict_.lookup("J0"))
                         )
    ),
    theta_
    (
        dimensionedScalar("theta",
                          dimensionSet(0, 0, 0, 0, 0),
                          readScalar(propsDict_.lookup("theta"))
                         )
    ),

    dParticle_
    (
        dimensionedScalar("dParticle",
                          dimensionSet(0, 1, 0, 0, 0),
                          readScalar(propsDict_.lookup("dParticle"))
                         )
    ),
    N_prim_
    (
        dimensionedScalar("NPrim",
                          dimensionSet(0, 0, 0, 0, 0),
                          readScalar(propsDict_.lookup("NPrim"))
                         )

    ),
    cellId2Report_(0)

{
    if (propsDict_.found("verbose")) verbose_=true;
    localThetaRad_  = theta_.value() * 3.14159265359 / 180;

    if (propsDict_.found("cellId2Report")) cellId2Report_=int(readScalar(propsDict_.lookup
("cellId2Report")));
}



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

nucleation::~nucleation()
{}
```

```

// * * * * * * * * * * * * * * * * Member Fct  * * * * * * * * * * * * * * * //
inline void nucleation::update() const
{
//    Info << "Computing source for " << iMoment << "th moment. Will use "
//         << iMoment-1+r_ << "th moment as the base" << endl;

    //Compute local properties and nucleation rate
    double localT_      = 0.0;
    double localSigma_  = 0.0;
    double localRhoLiq_ = 0.0;
    double localS_      = 0.0;
    double localRStar   = 0.0;
    double localddInit  = 0.0;
    double localGhet    = 0.0;
    double localJ_      = 0.0;
    double hEvap_       = 0.0;
    double localMassS_  = 0.0;
    double localHeatS_  = 0.0;


    forAll(T_.internalField(),cellI)
    {
        localT_     = T_.internalField()[cellI];
        localS_     = S_.internalField()[cellI];



        localSigma_ = liquids_.sigma(pRef_.value(), localT_, xLiquid_);
        localRhoLiq_= liquids_.rho  (pRef_.value(), localT_, xLiquid_);


        localRStar         =  2.0   //this is the critical radius of the droplets
                            * localSigma_ * MWVapor_
                            / (
                                 localRhoLiq_*Rgas_.value() * localT_ * log(max(1.0,localS_))
                               + 1e-10
                              );

       localddInit      = pow  //compute the radius of the initial droplet
                             (
                                  localRStar * localRStar * localRStar
                               +dParticle_.value() * dParticle_.value() * dParticle_.value() / 8.0
                              , 0.3333333333333333333333
                             );

        localGhet          =  4.188790205    // 4/3*pi
                            * localRStar*localRStar
                            * localSigma_
                            * fg(localRStar);


        //Save to fields
        rStar_.internalField()[cellI] = localddInit ;
        J_.internalField()[cellI] = J_0_.value()
                                  * exp(
                                         -1.0* localGhet / localT_ / kB_.value()
                                       );

        localJ_     = J_.internalField()[cellI];

        //Update mass and enthalpy
        massSource_.internalField()[cellI] =  J_.internalField()[cellI]
                                           * localRStar * localRStar * localRStar
                                           * 4.188790205    // 4/3*pi
                                           * localRhoLiq_;
        /*massSource_.internalField()[cellI] =  0;*/

        localMassS_ = massSource_.internalField()[cellI];

        hEvap_ = liquids_.hl(pRef_.value(), localT_, xLiquid_);
        energySource_.internalField()[cellI] =  massSource_.internalField()[cellI]
                                             *  hEvap_;

        localHeatS_ = energySource_.internalField()[cellI];


        //Print stats to file
        if(verbose_ && cellI == cellId2Report_)
        {
            Pout << "nucleation[" << cellId2Report_
                 << "]: T S sigma rhoLiq rStar dInit Ghet J hEvap massSource heatSource:  "
             << localT_ << "   "
             << localS_ << "   "
             << localSigma_ << "   "
             << localRhoLiq_ << "   "
             << localRStar << "   "
             << localddInit << "   "
             << localGhet << "   "
             << localJ_ << "   "
             << hEvap_ << "   "
             << localMassS_ << "   "
             << localHeatS_ << "   " << endl;
        }

    }

    isUpToDate_ = true;

}

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

inline volScalarField& nucleation::source(autoPtr<transportModel>*  transportModels, int iMoment)
const
{
    if( iMoment==0 )
    {

      double timeStep = particleCloud_.mesh().time().deltaT().value() + 1e-99;
      forAll(source_.internalField(),cellI)
      {
          //Correct the nucleation rate to bound m_0 to N_prim_
          double m_0Old = transportModels[0]->m().internalField()[cellI];
          if(  (m_0Old + J_.internalField()[cellI] * timeStep) > N_prim_.value())
          {
                double Jold = J_.internalField()[cellI];
                J_.internalField()[cellI] = max( 0.0,
                                            ( N_prim_.value() - m_0Old ) / timeStep
                                          );   //bound by zero, i.e., always nucleation,

                //Update mass and heat exchange rate
                massSource_.internalField()[cellI]   *= J_.internalField()[cellI]
                                                      / max(Jold,1e-32);
                energySource_.internalField()[cellI] *= J_.internalField()[cellI]
                                                      / max(Jold,1e-32);
          }

          source_.internalField()[cellI]  = J_.internalField()[cellI];

       }
    }
    else    //source term for kth moment
    {
```

File: /home/mario/OpenFOAM/mario-2....el/simpleGrowth/simpleGrowth.C Page 1 of 6

```
/*---------------------------------------------------------------------------*\
License

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

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

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

    Copyright (C) 2014- Stefan Radl, TU Graz, Austria

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

#include "error.H"
#include "simpleGrowth.H"
#include "transportModel.H"
#include "qmomSolver.H"
#include "addToRunTimeSelectionTable.H"
#include <sstream>
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

namespace Foam
{

// * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //
defineTypeNameAndDebug(simpleGrowth, 0);

addToRunTimeSelectionTable
(
    physicalModel,
    simpleGrowth,
    dictionary
);

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

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

// Construct from components
simpleGrowth::simpleGrowth
(
    const dictionary& dict,
    qmomCloud& sm,
    int modelID
)
:
    physicalModel(dict,sm,modelID),
    propsDict_(dict.subDict(typeName + "Props")),
    verbose_(false),
    r_(readScalar(propsDict_.lookup("r"))), //just for testing, obsolete if useLocalG_ is used
    G_0_    //just for testing, obsolete if useLocalG_ is used
    (
        dimensionedScalar("G0",
                          dimensionSet(0, 0, -1, 0, 0),
                          readScalar(propsDict_.lookup("G0"))
                         )
    ),
    useLocalG_(false),
    G_
    (
```

File: /home/mario/OpenFOAM/mario-2....el/simpleGrowth/simpleGrowth.C Page 2 of 6

```
        IOobject
        (
           "Growth",
            sm.mesh().time().timeName(),
            sm.mesh(),
            IOobject::READ_IF_PRESENT,
            IOobject::AUTO_WRITE
        ),
        sm.mesh(),
        dimensionedScalar("zero",
                          dimensionSet(0, 0, -1, 0, 0),
                          scalar(0.0)
                         )
    ),
    SFieldName_(propsDict_.lookup("SName")),
    S_(sm.mesh().lookupObject<volScalarField> (SFieldName_)),
    YFieldName_(propsDict_.lookup("YName")),
    Y_(sm.mesh().lookupObject<volScalarField> (YFieldName_)),
    RhoGFieldName_(propsDict_.lookup("RhoGName")),
    RhoG_(sm.mesh().lookupObject<volScalarField> (RhoGFieldName_)),
    DiffFieldName_(propsDict_.lookup("DiffName")),
    diff_(sm.mesh().lookupObject<volScalarField> (DiffFieldName_)),
    requiresPDF_(false),
    massSourceIsSet_(false),
    limitGrowthFactor_(-1.0)
{
    if (propsDict_.found("verbose")) verbose_      = readBool(propsDict_.lookup("verbose"));
    if (propsDict_.found("useQmom")) requiresPDF_  = readBool(propsDict_.lookup("useQmom"));

    if (propsDict_.found("useLocalG"))
    {
       useLocalG_=true;
       r_ = -1;
       printf("simpleGrowth: using local growth rate \n");
    }

    if (propsDict_.found("limitGrowthFactor"))
    {
       limitGrowthFactor_=readScalar(propsDict_.lookup("limitGrowthFactor"));
       if(limitGrowthFactor_<1.1)
           FatalError << "limitGrowthFactor<1.1: This will limit growth very much. Choose this 
parameter > 1.1 !" << abort(FatalError) << endl;
       printf("simpleGrowth: using non-standard limitGrowthFactorof %g \n",limitGrowthFactor_);
    }

    //Check r
    if(r_==-1)
    {
        printf("using r=-1 for simpleGrowth model \n");
    }
    else if(r_==0)
        printf("using r=0 for simpleGrowth model \n");
    else if(r_==1)
        printf("using r=1 for simpleGrowth model \n");
    else
    {
        requiresPDF_ = true;
    }

    if(requiresPDF_)
        Info << "simpleGrowth will use QMOM." << endl;

}


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

```

File: /home/mario/OpenFOAM/mario-2....el/simpleGrowth/simpleGrowth.C Page 3 of 6

```
simpleGrowth::~simpleGrowth()
{}

// * * * * * * * * * * * * * * * * * Member Fct  * * * * * * * * * * * * * * * //
inline void simpleGrowth::update() const
{

    //Compute local properties and nucleation rate
    double localT_      = 0.0;
    double localRhoLiq_ = 0.0;
    double localRhoG_   = 0.0;
    double localDiffG_  = 0.0;

    double localS_      = 0.0;
    double localY_      = 0.0;
    double localBm_     = 0.0;  //spalding mass transfer number
    double localYsf_    = 0.0;  //equilibrium mass fraction (directly above droplet surface)

    forAll(T_.internalField(),cellI)
    {
        localT_     = T_.internalField()[cellI];
        localS_     = S_.internalField()[cellI];
        localY_     = Y_.internalField()[cellI];
        localRhoG_  = RhoG_.internalField()[cellI];
        localDiffG_ = diff_.internalField()[cellI];

        localRhoLiq_= liquids_.rho  (pRef_.value(), localT_, xLiquid_);


        //Using the definition of the Saturation, assuming thermal equilibrium of gas and droplets
        //as well as the dimensionless mass fraction,
        //we can compute the local saturation mass fraction from Y and S:
        //S = Yi / localYsf
        //Yi = YSat * Y , here YSat is the reference saturation mass fraction @ the evaporator
        // --> localYsf = YSat * Y / S
        localYsf_ = YSat_ * localY_
                  / fmax(1e-12, localS_);
        if( fabs(1.0 - localYsf_) > 1e-12)
          localBm_ = (YSat_*localY_ - localYsf_) / (1.0 - localYsf_);
        else
          localBm_ = 0.0; //in case we have S = 1, there is no mass transfer to the droplet. this
is to avoid division by zero

        //Save to fields, this is G * \xi    !!
        G_.internalField()[cellI] = 4.0 //2*Sh
                                   *localRhoG_ / localRhoLiq_
                                   *localDiffG_
                                   *log( fmax(1e-12,1.0 + localBm_) );

    }
    //printf("min/max G_: %.5f / %.5f \n", min(G_).value(), max(G_).value()); //TODO: also check
other variables
    if(verbose_)
    {
        printf("localT_ = %.5f \n", localT_);
        printf("localS_ = %.5f \n", localS_);
        printf("localY_ = %.5f \n", localY_);
        printf("localRhoG_ = %.5f \n", localRhoG_);
        printf("localDiffG_ = %.5f \n", localDiffG_);
        printf("localRhoLiq_ = %.5f \n", localRhoLiq_);
        printf("YSat = %.5f \n", YSat_);
    }

    isUpToDate_ = true;
}


```

```

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
inline volScalarField& simpleGrowth::source(autoPtr<transportModel>*  transportModels, int iMoment)
const
{
//    Info << "Computing source for " << iMoment << "th moment. Will use "
//         << iMoment-1+r_ << "th moment as the base" << endl;

  //Growth cannot change m0
  if(iMoment==0)
  {
    source_ *= 0.0;
    return source_;
  }

  //Moment used in the source expression
  int k = iMoment-1+r_;

  // *** Use QMOM ***
  if(requiresPDF_)
  {
      //Set the required moment exponent for the source term


      //Set the source
      if(useLocalG_)
      {
       source_ =   G_
                 * double(iMoment)
                 * particleCloud_.qmomS().reconstructMoment( transportModels, k );
      }
      else
      {
        source_  = G_0_
                 * double(iMoment)
                 * particleCloud_.qmomS().reconstructMoment( transportModels, k );
      }
  }
  // *** Use MOM ***
  else
  {
    if( (k) >= 0)
    {
      if(useLocalG_)
      {
        source_  = G_
               * double(iMoment)
               * transportModels[k]->m();
      }
      else
      {
        source_  = G_0_
               * double(iMoment)
               * transportModels[k]->m();
      }
    }
    else    //return zero growth rate
    {
      source_ *= 0.0;
      Info << "simpleGrowth: WARNING: Neglecting source term for " << iMoment << "-th moment
because MOM is used." << endl;
    }
  }

  //Limit the growth rate - just do for moment 1 (to fix G_ for other moments)
  if(limitGrowthFactor_>0 && iMoment==1)
  {
      double maxIncr_ = limitGrowthFactor_-1.0;
      double deltaT = T_.mesh().time().deltaT().value();
      double sourceError(0.0);
      double tooMuchFactor_(0.0);

      forAll(T_.internalField(),cellI) //loop all cells
      {
          if( (source_[cellI] * deltaT) > (transportModels[iMoment]->m()[cellI]*maxIncr_ ) )
          {
              tooMuchFactor_ = source_[cellI]
                             / ( maxIncr_*transportModels[iMoment]->m()[cellI] / deltaT )
                             + 1e-64;

              //Scale source and growth rate
              source_[cellI]/= tooMuchFactor_;
              sourceError   += G_[cellI]*(1.0-1.0/tooMuchFactor_);
              G_[cellI]     /= tooMuchFactor_;

              if(verbose_)
                Pout << "tooMuchFactor: " << tooMuchFactor_ << ". Resetting G[" << cellI << "] to " << G_[cellI] << endl;
          }
      }
      if(verbose_ && sourceError>0.0)
          Pout << "sourceError: " << sourceError << endl;
  }


  return source_;
}

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

inline volScalarField& simpleGrowth::massSource(autoPtr<transportModel>*  transportModels) const
{

  //Set the required moment exponent for the source term, same as for the third moment!
  int k = 3 - 1 + r_;

  // *** Use QMOM ***
  if(requiresPDF_)
  {
      //Set the source
      if(useLocalG_)
      {
       massSource_.internalField() =   G_
                 * 1.570796327 //3*pi/6
                 * particleCloud_.qmomS().reconstructMoment( transportModels, k );
      }
      else
      {
        massSource_.internalField()  = G_0_
                 * 1.570796327 //3*pi/6
                 * particleCloud_.qmomS().reconstructMoment( transportModels, k );
      }
  }
  // *** Use MOM ***
  else
  {
      if(useLocalG_)
      {
        massSource_.internalField()  = G_
                 * 1.570796327 //3*pi/6
               * transportModels[k]->m();
      }
      else
      {
        massSource_.internalField()  = G_0_
                 * 1.570796327 //3*pi/6
               * transportModels[k]->m();
      }
  }

  //Multiply with local liquid density
  double localT_      = 0.0;
  double localRhoLiq_ = 0.0;
  forAll(T_.internalField(),cellI)
  {
        localT_     = T_.internalField()[cellI];
        localRhoLiq_= liquids_.rho  (pRef_.value(), localT_, xLiquid_);

        massSource_.internalField()[cellI] *= localRhoLiq_;
  }


  massSourceIsSet_ = true;
  return massSource_;
}

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

inline volScalarField& simpleGrowth::energySource(autoPtr<transportModel>*  transportModels) const
{

  //Requires that mass source is called before!
  if(!massSourceIsSet_)
    FatalError << "Mass source is not set in simpleGrowth::energySource. Fail!" << abort
(FatalError) << endl;


  double localT_      = 0.0;
  double hEvap_       = 0.0;

  forAll(T_.internalField(),cellI)
  {
        localT_     = T_.internalField()[cellI];
        hEvap_      = liquids_.hl(pRef_.value(), localT_, xLiquid_);

        energySource_.internalField()[cellI] = massSource_.internalField()[cellI] * hEvap_;
  }

  return energySource_;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

} // End namespace Foam

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

File: /home/mario/OpenFOAM/mario-2....PhysSpace/diffusionPhysSpace.C Page 1 of 4

```
/*---------------------------------------------------------------------------*\
License

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

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

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

    Copyright (C) 2014- Stefan Radl, TU Graz, Austria

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

#include "error.H"
#include "diffusionPhysSpace.H"
#include "transportModel.H"
#include "qmomSolver.H"
#include "addToRunTimeSelectionTable.H"
#include <sstream>
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

namespace Foam
{

// * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //
defineTypeNameAndDebug(diffusionPhysSpace, 0);

addToRunTimeSelectionTable
(
    physicalModel,
    diffusionPhysSpace,
    dictionary
);

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

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

// Construct from components
diffusionPhysSpace::diffusionPhysSpace
(
    const dictionary& dict,
    qmomCloud& sm,
    int modelID
)
:
    physicalModel(dict,sm,modelID),
    propsDict_(dict.subDict(typeName + "Props")),
    verbose_(false),
    gamma_0_    //just for testing, obsolete if useLocalDiff_ is used
    (
        dimensionedScalar("gamma0",
                          dimensionSet(0, 2, -1, 0, 0),
                          readScalar(propsDict_.lookup("gamma0"))
                         )
    ),
    dynViscosity_(1e-5),
    gamma_
    (
        IOobject
        (
           "diffParticle",
            sm.mesh().time().timeName(),
            sm.mesh(),
            IOobject::READ_IF_PRESENT,
            IOobject::NO_WRITE
        ),
        sm.mesh(),
        dimensionedScalar("zero",
                          dimensionSet(0, 2, -1, 0, 0),
                          scalar(0.0)
                         )
    ),
    useLocalDiff_(false),
    requiresPDF_(false)
{
    if (propsDict_.found("verbose")) verbose_      = readBool(propsDict_.lookup("verbose"));
    if (propsDict_.found("useQmom")) requiresPDF_  = readBool(propsDict_.lookup("useQmom"));

    dynViscosity_ = readScalar(propsDict_.lookup("dynViscosity"));

    if (propsDict_.found("useLocalDiff"))
    {
       useLocalDiff_=true;
       printf("using local diffusion rate \n");
    }

    if(requiresPDF_)
        Info << "diffusionPhysSpace will use QMOM." << endl;

}


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

diffusionPhysSpace::~diffusionPhysSpace()
{}

// * * * * * * * * * * * * * * * * Member Fct  * * * * * * * * * * * * * * * //
inline void diffusionPhysSpace::update() const
{

    //Compute local properties and nucleation rate
    double localT_      = 0.0;
    double invThreePiViscosity = 1.0
                               / (9.424777961 * dynViscosity_ + 1e-32);

    forAll(T_.internalField(),cellI)
    {
        localT_     = T_.internalField()[cellI];

        //Save to fields, this is G * \xi    !!
        gamma_.internalField()[cellI] = kB_.value() * localT_ * invThreePiViscosity ;

        if(verbose_)
        {
            printf("localT_ = %.5f \n", localT_);
            printf("gamma_  = %.5g \n", gamma_.internalField()[cellI]);
        }

    }

    isUpToDate_ = true;
}



```

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
inline volScalarField& diffusionPhysSpace::source(autoPtr<transportModel>*  transportModels, int
iMoment) const
{
//    Info << "Computing source for " << iMoment << "th moment. Will use "
//         << iMoment-1+r_ << "th moment as the base" << endl;

  // *** Use QMOM ***
  if(requiresPDF_)
  {
      //Set the source
    if( (iMoment-1) >= 0)
    {
      if(useLocalDiff_)
      {
       source_ =   fvc::laplacian(gamma_, transportModels[iMoment-1]->m(),"laplacian(Dp,m)");
      }
      else
      {
        source_  = fvc::laplacian(gamma_0_, transportModels[iMoment-1]->m(),"laplacian(Dp,m)");
      }
    }
    else    //return source based on reconstructed moment set
    {
      if(useLocalDiff_)
      {
          source_   = fvc::laplacian(gamma_,
                                     particleCloud_.qmomS().reconstructMoment( transportModels,
iMoment-1 ),
                                     "laplacian(Dp,m)"
                                    );
      }
      else
      {
          source_   = fvc::laplacian(gamma_0_,
                                     particleCloud_.qmomS().reconstructMoment( transportModels,
iMoment-1 ),
                                     "laplacian(Dp,m)"
                                    );
      }
    }
  }
  // *** Use MOM ***
  else
  {
    if( (iMoment-1) >= 0)
    {
      if(useLocalDiff_)
      {
       source_ =   fvc::laplacian(gamma_, transportModels[iMoment-1]->m(),"laplacian(Dp,m)");
      }
      else
      {
        source_  = fvc::laplacian(gamma_0_, transportModels[iMoment-1]->m(),"laplacian(Dp,m)");
      }
    }
    else    //return zero growth rate
    {
      source_ *= 0.0;
      Info << "diffusionPhysSpace: WARNING: Neglecting source term for 0-th moment." << endl;
    }
  }

  return source_;
}


```

File: /home/mario/OpenFOAM/mario-2....PhysSpace/diffusionPhysSpace.C Page 4 of 4

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

} // End namespace Foam

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

File: /home/mario/OpenFOAM/mario-2....Solver/qmomSolver/qmomSolver.C Page 1 of 3

```
/*---------------------------------------------------------------------------*\
License

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

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

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

    Copyright (C) 2014- Stefan Radl, TU Graz, Austria


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

#include "error.H"
#include "qmomSolver.H"
#include "transportModel.H"

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

namespace Foam
{

//Helper function
inline string intToString (int a)
{
    std::ostringstream temp;
    temp<<a;
    return temp.str();
}

// * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //

defineTypeNameAndDebug(qmomSolver, 0);

defineRunTimeSelectionTable(qmomSolver, dictionary);

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

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

// Construct from components
qmomSolver::qmomSolver
(
    const dictionary& dict,
    qmomCloud& aCloud
)
:
    dict_(dict),
    particleCloud_(aCloud),
    N_(particleCloud_.nrtransportModels() / 2),
    nodes_(N_),
    weights_(N_)
{
  //Create the volScalarFields if required
  if(particleCloud_.requiresPDF())
  {

    if( N_ < 2)
    {
```

File: /home/mario/OpenFOAM/mario-2....Solver/qmomSolver/qmomSolver.C Page 2 of 3

```
    Info   << "You use " << N_ << " nodes and " << particleCloud_.nrtransportModels() << " moments." << endl;
    Info   << "WARNING: The solver will skip the reconstruction of nodes, and approximate the size distribution with a single diameter!" <<  endl;
  }

  if( N_ > 3)
    FatalError << "You are using more than 3 nodes, which requires a more complex eigenvalue/eigenvector computation. The solver cannot handle this. Use less moments (and nodes)." <<  abort(FatalError);


  if( particleCloud_.nrtransportModels() < (2*N_) )
    FatalError << "Too few moment transport equations. Add more transportModels!" <<  abort(FatalError);

  if( particleCloud_.nrtransportModels() > (2*N_) )
    FatalError << "Too much moment transport equations. Remove one or more transportModel, such that the number of moment equations matches the 2 * the number of nodes!" <<  abort(FatalError);


  for (int i=0; i<N_; i++)
  {
    nodes_[i].reset
    (
      new volScalarField
         (   IOobject
          (
            "node"+intToString(i),
            particleCloud_.mesh().time().timeName(),
            particleCloud_.mesh(),
            IOobject::READ_IF_PRESENT,
            IOobject::AUTO_WRITE
          ),
          particleCloud_.mesh()
         )
    );

    weights_[i].reset
    (
      new volScalarField
         (   IOobject
          (
            "weight"+intToString(i),
            particleCloud_.mesh().time().timeName(),
            particleCloud_.mesh(),
            IOobject::READ_IF_PRESENT,
            IOobject::AUTO_WRITE
          ),
          particleCloud_.mesh()
         )
    );


  }
 }
}


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

qmomSolver::~qmomSolver()
{
}

// * * * * * * * * * * * * * * public Member Functions  * * * * * * * * * * * * * //
const volScalarField qmomSolver::reconstructMoment(autoPtr<transportModel>*  transportModels, int
i) const
{
    tmp<volScalarField> myMoment(0.0 * transportModels[0]->m()); //initialize with zero

    for(int iNode=0; iNode < N_; iNode++)
    {
            myMoment  =  myMoment
                        +(
                             weight(iNode)
                           * pow( node(iNode) + scalar(1e-64), i )
                         );
    }

    myMoment =  myMoment * transportModels[0]->m(); //Multiply with 0-th moment (i.e., the
concentration)

    return myMoment;
}

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

} // End namespace Foam

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

```
/*---------------------------------------------------------------------------*\
License

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

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

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

    Copyright (C) 2014- Stefan Radl, TU Graz, Austria

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

#include "error.H"

#include "transportModel.H"
#include "productDifferenceQMOMSol.H"
#include "addToRunTimeSelectionTable.H"

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

namespace Foam
{

#define SMALLNUMBER 1e-30
#define VSMALLNUMBER 1e-100

// * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //

defineTypeNameAndDebug(productDifferenceQMOMSol, 0);

addToRunTimeSelectionTable
(
    qmomSolver,
    productDifferenceQMOMSol,
    dictionary
);


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

// Construct from components
productDifferenceQMOMSol::productDifferenceQMOMSol
(
    const dictionary& dict,
    qmomCloud& aCloud
)
:
    qmomSolver(dict,aCloud),
    propsDict_(dict.subDict(typeName + "Props")),
    alphaMin_(0.0),
    alphaLimited_(0),
    P_(2*N_+1),
    jacobi_(N_),
    mCurr_( aCloud.nrtransportModels() ),
    nodesCurr_(N_),
    weightsCurr_(N_),
    bkk_(aCloud.nrtransportModels()),
    bk_(aCloud.nrtransportModels()),
    d_(aCloud.nrtransportModels()),
```

File: /home/mario/OpenFOAM/mario-2....Sol/productDifferenceQMOMSol.C Page 2 of 11

```
    cos_quad_alfa_(aCloud.nrtransportModels()),
    verbose_(false),
    useCorrector_(false),
    iter_max_(100),
    cellIDVerbose_(0),
    monodisperseCorrFactor_(0.01),
    roundPrecision_(1e-8),
    computeEigenValues_(true),
    useEqualWeights_(false),
    minConcentration_(1e-6),
    haveJustOneNode_(false)
{

    simpleMeanDiameterId_[0] = 0;
    simpleMeanDiameterId_[1] = 1;

    //check settings
    if (propsDict_.found("verbose"))        verbose_=readBool(propsDict_.lookup("verbose"));
    if (propsDict_.found("useCorrector"))   useCorrector_=readBool(propsDict_.lookup
("useCorrector"));
    if (propsDict_.found("iter_max"))       iter_max_=readScalar(propsDict_.lookup("iter_max"));
    if (propsDict_.found("cellIDVerbose"))  cellIDVerbose_=readScalar(propsDict_.lookup
("cellIDVerbose"));
    if (propsDict_.found("monodisperseCorrFactor")) monodisperseCorrFactor_=readScalar
(propsDict_.lookup("monodisperseCorrFactor"));
    if (propsDict_.found("roundPrecision"))     roundPrecision_     = readScalar(propsDict_.lookup
("roundPrecision"));
    if (propsDict_.found("computeEigenValues")) computeEigenValues_ = readBool(propsDict_.lookup
("computeEigenValues"));
    if (propsDict_.found("useEqualWeights"))
    {
        useEqualWeights_    = readBool(propsDict_.lookup("useEqualWeights"));
        if(useEqualWeights_)
            Info << "WARNING: Will assign equal weights to each computed node (which is > " <<
SMALLNUMBER << ")." << endl;
            Info << "...This might be inaccurate (but stable)!" << endl;
    }
    if ( propsDict_.found("simpleMeanDiameterId0") && propsDict_.found("simpleMeanDiameterId1"))
    {
        simpleMeanDiameterId_[0]  = int(readScalar(propsDict_.lookup("simpleMeanDiameterId0")));
        simpleMeanDiameterId_[1]  = int(readScalar(propsDict_.lookup("simpleMeanDiameterId1")));
        Info << "Have found simpleMeanDiameterIds: " << simpleMeanDiameterId_[0] << " and "  <<
simpleMeanDiameterId_[1] << endl;
        Info << "...will divide moment[" << simpleMeanDiameterId_[1]
             << "] with moment["  << simpleMeanDiameterId_[0]
             << "] to approximate the distribution with a single node in case this is necessary."
             << endl;
    }

    if (propsDict_.found("minConcentration"))   minConcentration_   = readScalar(propsDict_.lookup
("minConcentration"));

    Info << "WARNING: Will not attempt to reconstruct nodes below m[0] = " << minConcentration_ <<
endl;
    if(!computeEigenValues_)
        Info << "WARNING: PD algorithmus will not use the eigenvalues, but just guess them. " <<
endl;

    //Allocate and reset values of temp arrays
    forAll(P_, i)
    {
        P_[i].setSize(2*N_+1);
        for(int j=0; j < P_[i].size(); j++)
        {
            P_[i][j] = 0.0;
        }
    }

    forAll(jacobi_, i)
    {
        jacobi_[i].setSize(N_);
        for(int j=0; j < jacobi_[i].size(); j++)
        {
            jacobi_[i][j] = 0.0;
        }
    }

    forAll(mCurr_, i)
    {
        mCurr_[i] = 0.0;
    }

    int nodeCount_=0;
    forAll(nodesCurr_,i)
    {
        nodesCurr_[i]   = 0.0;
        weightsCurr_[i] = 0.0;
        nodeCount_++;
    }
    if(nodeCount_<2)
    {
        haveJustOneNode_ = true;
        Info << "PD algorithm: made settings for " << nodeCount_ << " node(s)." << endl;
        Info << "This is less than two, and hence the algorithm will do a simplified calculation"
<< endl;
    }
    //initialize only if corrector is used
    if(!useCorrector_)
        return;

    forAll(d_, i)
    {
        d_[i].setSize( particleCloud_.nrtransportModels() );
        for(int j=0; j < d_[i].size(); j++)
        {
            d_[i][j]  = 0.0;
        }
    }

    forAll(bkk_, i)
    {
        cos_quad_alfa_[i] = 0.0;

        bkk_[i].setSize(bkk_.size());
        bk_[i].setSize(bk_.size());


        for(int j=0; j < bkk_[i].size(); j++)
        {
            bkk_[i][j] = 0.0;
            bk_[i][j]  = 0.0;
        }
    }
    if(verbose_)
    {
        Info << "size P_: " << P_.size() << " , " << P_[0].size() << endl;
        Info << "size jacobi_: " << jacobi_.size() << " , " << jacobi_[0].size() << endl;
        Info << "size bk_: " << bk_.size() << " , " << bk_[0].size() << endl;
    }

}


// * * * * * * * * * * * * * * * * Destructor  * * * * * * * * * * * * * * * //
```

```

productDifferenceQMOMSol::~productDifferenceQMOMSol()
{}


// * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * //
void productDifferenceQMOMSol::correctMoments(scalarList& m, int & totalIt ) const
{
    int    k_star  = 0;
    double lnck    = 0.0;

    if(!useCorrector_)
        return;

    //reset matrices
    forAll(bkk_, i)
    {
        for(int j=0; j < bkk_[i].size(); j++)
        {
            bkk_[i][j] = 0.0;
            bk_[i][j]  = 0.0;
        }
    }

    //define the unitary corretion matrix, index starts at 0!
    forAll(bkk_, k)
    {
        for(int i=0; i < bkk_[k].size(); i++)
        {
            if(i==k)
                bkk_[i][0] = 1.0;
            else
                bkk_[i][0] = 0.0;

        }


        for(int j=1; j < 4; j++)
            for(int i=0; i < (bkk_[k].size()-(j+1)+1); i++)
                    bkk_[i][j] = bkk_[i+1][j-1]
                               - bkk_[i][j-1];

        for(int i=0; i < bkk_[k].size(); i++)
            bk_[k][i] = bkk_[i][3];
    }

    bool check_corr = true;
    int  iter = 0;

    while(check_corr && iter<iter_max_)
    {
        // 1 - Check the need for correction
        check_corr = false;

        for(int i=0; i < bkk_.size(); i++)
            d_[i][0] = log(fmax(VSMALLNUMBER,m[i]));

        for(int j=1; j < (particleCloud_.nrtransportModels()+1); j++)
        {
            for(int i=0; i < (particleCloud_.nrtransportModels()-(j+1) + 1); i++)
            {
                d_[i][j] = d_[i+1][j-1]-d_[i][j-1];
                //Ueberpruefen, ob d_2 ein negatives Element enthaelt, wenn ja, dann wird
               //check_corr=1 gesetzt, d.h. es wird weiter korrigiert
                if (j==2 && d_[i][j]<0.0)
                   check_corr = true;

```

File: /home/mario/OpenFOAM/mario-2....Sol/productDifferenceQMOMSol.C Page 5 of 11

```
            }
        }

        // 2 - Correct the moment
        if(check_corr)
        {
            iter++;
            totalIt++;
            k_star = 0; //set starting index

            for(int k=0; k<particleCloud_.nrtransportModels(); k++)
            {
                //Compute scalar products
                double tmpCos1 = 0.0;
                double tmpCos2 = 0.0;
                double tmpCos3 = 0.0;
                for(int sumI=0; sumI < bk_[k].size(); sumI++)
                {
                    tmpCos1 += bk_[k][sumI]*d_[sumI][3];
                    tmpCos2 += bk_[k][sumI]*bk_[k][sumI];
                    tmpCos3 += d_[sumI][3]*d_[sumI][3];
                }

                //compute k_star
                double tmpCos4 =      tmpCos1
                                      /
                                       (
                                            sqrt(tmpCos2 * tmpCos3)
                                          + SMALLNUMBER
                                       );
                cos_quad_alfa_[k] = tmpCos4 * tmpCos4;

                if( cos_quad_alfa_[k] >= cos_quad_alfa_[k_star] )
                    k_star = k;
            }

            //Actually correct
//check  [4]
            double tmpCos1 = 0.0;
            double tmpCos2 = 0.0;
            for(int sumI=0; sumI < bk_[k_star].size(); sumI++)

            {
                tmpCos1 -= bk_[k_star][sumI]*d_[sumI][3];
                tmpCos2 += bk_[k_star][sumI]*bk_[k_star][sumI];
            }

            lnck =   tmpCos1
                    /
                    ( tmpCos2 + SMALLNUMBER );

            m[k_star] = exp(lnck)*m[k_star];
        }

        if(verbose_)
        {
            Info << "iter: " << iter << " d: " << d_ ;
//            Info << "bkk: " << bkk_ << endl;
//            Info << "bk: " << bk_ << endl;
//            Info << "m: " << m << endl;
            Info << "check_corr: " << check_corr << endl;
            Info << "lnck: " << lnck << " k_star: " << k_star << " m[k_star]: " <<  m[k_star] <<
endl  << endl;

        }

    }
```

```


}

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

void productDifferenceQMOMSol::solve(autoPtr<transportModel>*  transportModels) const
{
    if(!particleCloud_.requiresPDF())
        return;

    int totalCorrIt = 0;
    forAll(particleCloud_.mesh().cells(), cellI)
    {

        if(
             (transportModels[0]->m()[cellI] <= minConcentration_)
           ||(transportModels[1]->m()[cellI] <= VSMALLNUMBER)
          )  //skip solver if concentration or first moment is zero
        {
            for(int iNode=0; iNode < N_; iNode++)
            {
                nodes_[iNode]->internalField()[cellI]   = 0.0;
                weights_[iNode]->internalField()[cellI] = 0.0;
            }
            continue;
        }

        //Fill local moments into container
        forAll(mCurr_, i)
        {
            mCurr_[i] = transportModels[i]->m()[cellI];
        }

        if(haveJustOneNode_)
        {
            computeNodesSimple(nodesCurr_, weightsCurr_, mCurr_);
        }
        else
        {
            //Correct Moments if necessary
            correctMoments(mCurr_,totalCorrIt);

            //Compute the nodes
            computeJacobi(jacobi_, P_, mCurr_);
            computeNodes(nodesCurr_, weightsCurr_, jacobi_, mCurr_);
        }


        //Push computed values into fields
        for(int iNode=0; iNode < N_; iNode++)
        {
            nodes_[iNode]->internalField()[cellI]   = nodesCurr_[iNode];
            weights_[iNode]->internalField()[cellI] = weightsCurr_[iNode];
        }

        //report to screen
        if(verbose_ && cellI==cellIDVerbose_)
        {
               Info << "cellI:  " << cellI << " m: " << mCurr_ << endl;
               Info << "P:      " << P_ << endl;
               Info << "jac:    " << jacobi_ << endl;
               Info << "nodes:  " << nodesCurr_ << endl;
               Info << "weights:" << weightsCurr_ << endl;
        }
    }

    if(verbose_)
    {

           Info << "totalCorrIt:  " << totalCorrIt << endl;
    }


}

// ************************************************************************************
inline double productDifferenceQMOMSol::subtractRoundoff(double a, double b, double precision) const
{
    double result = a - b;
    if( std::abs(result) < (precision*std::abs(a)) )
        result = 0.0;
    return result;
}

// ************************************************************************************
void productDifferenceQMOMSol::computeJacobi( scalarListList& jacobi, scalarListList& P, scalarList
m ) const
{

//    Info << "productDifferenceQMOMSol - Computing "
//           << N_
//           <<" nodes for moments " << m << endl;
    //Some Interesting computations!
//    printf("m[i]-m[1]^i: %.15g %.15g %.15g %.15g %.15g %.15g \n",
//                m[0],
//                m[1]-m[1],
//                m[2]-m[1]*m[1],
//                m[3]-m[1]*m[1]*m[1],
//                m[4]-m[1]*m[1]*m[1]*m[1],
//                m[5]-m[1]*m[1]*m[1]*m[1]*m[1]);

    //Fill the P_ matrix, indices start with 0!
    P[0][0] = 1.0;

    for(int i=1; i<(2*N_+1); i++)   //1st column
        P[i][0] = 0.0;

    for(int i=0; i<(2*N_); i++)     //2nd column
       P[i][1] = pow(-1,i) * m[i];

    for(int j=2; j<(2*N_+1); j++)   //3rd+ column
       for(int i=0; i<(2*N_+1-j); i++)
       {
            P[i][j] =  subtractRoundoff(
                        P[0][j-1] * P[i+1][j-2],
                        P[0][j-2] * P[i+1][j-1],
                        roundPrecision_);

       }

    //Fill the zetas, indices start with 0!
    scalarList zeta(2*N_);
    zeta[0]=0.0;
    for(int i=1; i<(2*N_); i++)
    {
        if( P[0][i]*P[0][i-1] > 0 )
            zeta[i] =   P[0][i+1]
                    / ( P[0][i] * P[0][i-1]);
        else
        {
            zeta[i]=0;  //TODO: check

            if(m[1]<VSMALLNUMBER)
```

```
                FatalError << "1st moment is zero or very small!" << abort(FatalError) << endl;

            //Implementation of correction for mono-disperse systems
            if(i==3) //4-th moment
            {

                double z1  = m[0]*m[2] - m[1]*m[1];
                double eps = fmax(std::abs(z1)*monodisperseCorrFactor_, SMALLNUMBER);
                double z2  = m[1] * sqrt( (z1+eps) * (z1+eps) );
//                Pout << "i=3 - m[1]:" << m[1] << " P[0][i+1]: " << P[0][i+1] << " z1:" << z1 << "
eps: " << eps << " z2: " << z2 << endl;
                zeta[i]    = P[0][i+1]/z2;
            }


            if(i==4 || i==5) //5-th or 6-th moment
            {
                double z1(0),z2(0);
                if(i==4)  {
                    z1  = m[0]*m[2] - m[1]*m[1];
                    z2  = m[1]*m[3] - m[2]*m[2];
                }
                else  {
                    z1  = m[1]*m[3] - m[2]*m[2];
                    z2  = m[2]*( m[1]-m[3] )
                        + m[0]*( m[4]-m[2] );
                }

                if(z1>=0.0 && z2>0.0)   {
                    double eps1 = fmax(std::abs(z1)*monodisperseCorrFactor_,
                                       SMALLNUMBER);
                    z1 = sqrt((z1 + eps1)*(z1 + eps1));
                }
                else if(z2>=0.0 && z1>0.0)   {
                    double eps2 = fmax(std::abs(z2)*monodisperseCorrFactor_,
                                       SMALLNUMBER);
                    z2 = sqrt((z2 + eps2)*(z2 + eps2));
                }
//                else if(z1==0.0 && z2==0.0)    {
                else    {
                    z1 = SMALLNUMBER;
                    z2 = SMALLNUMBER;
                }

                if(i==4)
                    zeta[i]    =  P[0][i+1]/(m[0]*z1*z2);
                else
                    zeta[i]    =  P[0][i+1]/(m[0]*m[0]*z1*z1*z2);
            }


            //Prints
            if(verbose_)
                Pout << "m0*m2 - m1^2 = " << m[0]*m[2] - m[1]*m[1]
                     << ",  zeta[" << i << "]: " << zeta[i] << endl;

        }
    }

    //Fill coefficients for Jacobi
    scalarList a(N_);
    scalarList b(N_-1);
    for(int i=0; i<N_; i++)
    {
        a[i] = zeta[2*i+1]
             + zeta[2*i];
    }

    for(int i=0; i<(N_-1); i++)
    {
        b[i] = zeta[2*(i+1)]
             * zeta[2*i+1];
    }



    for(int i=0; i<(2*N_); i++)
        if(verbose_)

           Pout << "zeta[" << i << "]: " << zeta[i]  << endl;




        if(verbose_)

           Pout << " a1: " << a[1] << ",  a2: " << a[2] << ",  a3: " << a[3]
                << " b1: " << b[1] << ",  b2: " << b[2]<< endl;



    //Fill Jacobi
    for(int i=0; i<N_; i++)
        jacobi[i][i]=a[i];

    for(int i=0; i<(N_-1); i++)
    {
        jacobi[i][i+1] = -sqrt(fmax(0.0,b[i]));   //TODO: check!
        jacobi[i+1][i] = -sqrt(fmax(0.0,b[i]));   //TODO: check!
    }



}

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

void productDifferenceQMOMSol::computeNodesSimple(scalarList& nodes, scalarList& weights,
scalarList m) const
{
     nodes[0]    =  m[simpleMeanDiameterId_[1]]
                 / (m[simpleMeanDiameterId_[0]]+VSMALLNUMBER);
     weights[0]  = 1.0;
}

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

void productDifferenceQMOMSol::computeNodes(scalarList& nodes, scalarList& weights, scalarListList
jacobi, scalarList m) const
{


    //Convert scalarList to OpenFOAM tensor
    tensor myT = tensor::zero;
    myT.xx() = jacobi[0][0]; myT.xy() = jacobi[0][1];
    myT.yx() = jacobi[1][0]; myT.yy() = jacobi[1][1];

    //Initialize a with diagonal elements of Jacobi
    vector a = vector(myT.xx(), myT.yy(), 0.0);

    if(N_>2)
    {
       myT.xz() = jacobi[0][2];
       myT.yz() = jacobi[1][2];
```

```
        myT.zx() = jacobi[2][0];
        myT.zy() = jacobi[2][1];
        myT.zz() = jacobi[2][2];
        a[2] = myT.zz();
    }
    vector aJacobi = a; //Save in case it is used later

    //Compute eigenvalues & corresponding eigenvectors
    if(computeEigenValues_)
        a = eigenValues(myT);

    double weightSum = 0.0;
    for(int i=0; i<N_; i++)
    {
        if(N_==2)
            nodes[i] = fmax(VSMALLNUMBER,a[i+1]);   //negative nodes are meaningless
        else if (N_==3)
            nodes[i] = fmax(VSMALLNUMBER,a[i]);     //negative nodes are meaningless

        if(useEqualWeights_)
        {
            if(nodes[i] >= SMALLNUMBER)
                weights[i] = 1.; //Assign equal weight to each node, normalize later
            else
                weights[i] = 0.; //Assign equal weight to each node, normalize later
        }
        else
        {
            vector eigenVec = eigenVector(myT, nodes[i]);
            weights[i] = fmin(1.0,eigenVec[0]*eigenVec[0]*m[0]); //bound weight to be max 1
        }
        weightSum += weights[i];

    }

    //Correct Nodes and Weights in case Algorithm has computed wrong weights
    if( (weightSum<(1.0-1e-5)) || (weightSum>(1.0+1e-5)) )
    {
        if( weightSum>1e-3 )        //Just Re-normalize
        {
            for(int i=0; i<N_; i++)
            {
                weights[i] /= (weightSum+SMALLNUMBER); //Re-normalize
            }
            weightSum = 1.0;
        }
        else    //There is a serious problem with the weights, they are too small!
        {
            weightSum = 0.0;
            for(int i=0; i<N_; i++)
            {
               if(nodes[i]>VSMALLNUMBER) //We have a valid node
               {
                   weights[i] = 1.0;
                   weightSum += 1.0;
               }
               else
               {
                   nodes[i]   = 0.0;
                   weights[i] = 0.0;
               }
            }

            if( weightSum < 1.0)  //Re-normalization of weights also failed. Enforce simple
calculation using trace of Jacobi
            {
              a         = aJacobi;  //force use of Jacobi elements as eigenvalues
```

```
                weightSum = 0.0;
                for(int i=0; i<N_; i++)
                {
                  if(N_==2)
                      nodes[i] = fmax(VSMALLNUMBER,a[i+1]);   //negative nodes are meaningless
                  else if (N_==3)
                      nodes[i] = fmax(VSMALLNUMBER,a[i]);  //negative nodes are meaningless

                  vector eigenVec = eigenVector(myT, nodes[i]);
                  weights[i] = eigenVec[0]*eigenVec[0]*m[0];
                  weightSum += weights[i];
                }
              }

              //Final Checks
              for(int i=0; i<N_; i++)
                  weights[i] /= (weightSum+1e-64);

              if( weightSum < (1.0-1e-5))
              {
                  //Fall back and compute a node based on two moments
                  //e.g., m1 and m0
                  //in case we have a mimimum concentration,
                  //set to zero (no growth possible)
                  int iStart = 0;
                  if(m[0]>minConcentration_)
                  {
                      nodes[0]    =  m[simpleMeanDiameterId_[1]]
                                  / (m[simpleMeanDiameterId_[0]]+VSMALLNUMBER);
                      weights[0]  = 1.0;
                      iStart = 1;
                  }

                  for(int i=iStart; i<N_; i++)
                  {
                      nodes[i]   = 0.0;
                      weights[i] = 0.0;
                  }

                  if(verbose_)
                      Info << "****PD Algorithm Serious WARNING: Calculation of weights failed "
                           << "because of ill jacobi / eigenvector calculation. Using m[0] and m[1]
to estimate node[0]." << endl;
              }

              if(verbose_)
                  Info << "****Corrected Nodes: " << nodes << ", weights: " << weights << endl;
          }
      }
  }



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

  } // End namespace Foam

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

E.2 Main Files of the Pipe Flow Case

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1;

pipeLength_1 0.1;
pipeLength_2 0.160;
pipeLength_3 0.190;
pipeLength_4 0.243;
pipeLength_5 0.343;

pipeRadius 0.0023;
deltaY     0.00008; //for 0.1 diameter pipe use  0.0034905
deltaYM -0.00008;//for 0.1 diameter pipe use  0.0034905

resolutionRadial 30;
resolutionPipe_1  100;
resolutionPipe_2  200;
resolutionPipe_3  200;
resolutionPipe_4  200;
resolutionPipe_5  100;
refinementRadial 0.5;

//***********END OF USER INPUT*******************************

//explicitely specify name and type of error patch
//to avoid troubles with collapseEdges
defaultPatch
{
    name errorPatch;
    type   patch;
}

vertices
(
    (0 0 0)                                     //0
    ($pipeLength_1 0 0)                         //1
    (0              $deltaYM   $pipeRadius)     //2
    ($pipeLength_1  $deltaYM   $pipeRadius)     //3
    ($pipeLength_1  $deltaY    $pipeRadius)     //4
    (0              $deltaY    $pipeRadius)     //5
    ($pipeLength_2  $deltaY    $pipeRadius)     //6
    ($pipeLength_2  $deltaYM   $pipeRadius)     //7
    ($pipeLength_2  0 0)                //8
    ($pipeLength_3  $deltaY    $pipeRadius)             //9
    ($pipeLength_3  $deltaYM   $pipeRadius)             //10
    ($pipeLength_3  0 0)                //11
    ($pipeLength_4  $deltaY    $pipeRadius)     //12
    ($pipeLength_4  $deltaYM   $pipeRadius)     //13
    ($pipeLength_4  0 0)                //14
    ($pipeLength_5  $deltaY    $pipeRadius)     //15
    ($pipeLength_5  $deltaYM   $pipeRadius)     //16
    ($pipeLength_5  0 0)                //17
);
```

```

blocks
(
    hex (0 1 1 0 2 3 4 5) ($resolutionPipe_1 1 $resolutionRadial) simpleGrading (1 1
$refinementRadial)
    hex (1 8 8 1 3 7 6 4) ($resolutionPipe_2 1 $resolutionRadial) simpleGrading (1 1
$refinementRadial)
    hex (8 11 11 8 7 10 9 6) ($resolutionPipe_3 1 $resolutionRadial) simpleGrading (1 1
$refinementRadial)
    hex (11 14 14 11 10 13 12 9) ($resolutionPipe_4 1 $resolutionRadial) simpleGrading (1 1
$refinementRadial)
    hex (14 17 17 14 13 16 15 12) ($resolutionPipe_5 1 $resolutionRadial) simpleGrading (1 1
$refinementRadial)
);

edges
(
/*  arc 2 5 (0 0 $pipeRadius)*/
/*  arc 3 4 ($pipeLength_1 0 $pipeRadius)*/
);

patches
(
    patch
    inlet
    (
        (0 2 5 0)
    )

    patch
    outlet
    (
        (17 16 15 17)
    )

    wedge
    axi_symm-f
    (
        (0 1 3 2)
        (1 8 7 3)
        (8 11 10 7)
        (11 14 13 10)
        (14 17 16 13)
    )

    wedge
    axi_symm-r
    (
        (0 1 4 5)
        (1 8 6 4)
        (8 11 9 6)
        (11 14 12 9)
        (14 17 15 12)
    )

    wall
    fixedWalls_1
    (
        (2 3 4 5)
    )

    wall
    evaporator
    (
        (3 7 6 4)
    )

```

File: /home/mario/OpenFOAM/mario-2....onstant/polyMesh/blockMeshDict Page 3 of 3

```
    wall
    insulation
    (
        (7 10 9 6)
    )

    wall
    condenser
    (
        (10 13 12 9)
    )

    wall
    fixedWalls_5
    (
        (13 16 15 12)
    )

);

mergePatchPairs
(
);

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

File: /home/mario/Dropbox/DA_MarioS...M/2_Thesis/PDFs/qmomProperties Page 1 of 2

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      qmomProperties;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
velFieldName "U";
TFieldName   "T";

couplingModel; //main Switch to activate coupling to fluid equations

transportModels
(
    momentTransport //0
    momentTransport //1
    momentTransport //2
    momentTransport //3
//  momentTransport //4
//  momentTransport //5
);

physicalModels
(
    nucleation
    simpleGrowth
/*    diffusionPhysSpace //be cautioned when using this model! n*/
);

//qmomSolver  none; //productDifference;
qmomSolver  productDifference;



//Properties of Models and Solvers
momentTransportProps
{
/*    verbose     true;*/
    fieldName "m";
    Dp0        0;   //1e-8; //1e-9; //particle diffusivity. MUST set to zero in case advanced
diffusion modelID
                     //is used as a "physicalModels"
    limitMomentAbove    3;
    momentLimitFactor   5;

}

simpleGrowthProps
{
/*    verbose     true;*/
    useLocalG;      //Main switch to use local growth rate, comment out if G0 should be used

    G0          1.0;  //obsolete in case local growth rate (G0 is only used to test the code,
                        //see physical model "simpleGrowth")
    r           -1;    //obsolete in case local growth rate is used
    useQmom     true; //FORCES the usage of MOM or QMOM, so be careful what you do here!
    SName       "S";
    YName       "Y";
```

File: /home/mario/Dropbox/DA_MarioS...M/2_Thesis/PDFs/qmomProperties Page 2 of 2

```
    RhoGName    "rho";
    DiffName    "diffEff";

    limitGrowthFactor 1e8;

}


nucleationProps
{
/*    verbose     true;*/
    SName       "S";
    theta       0; //in grad
    dParticle   4e-8;
    J0          5.02655e14; //1e5;
    NPrim       1e10;       //NPrim limits the nucleation rate (and thus the particle number concentration) for every cell, not in the whole domain!!!

    cellId2Report 15001;    //id of the cell to report nucleation properties
}

productDifferenceProps
{
/*    verbose true;*/
    iter_max            10;
    useCorrector        false;  //Main Switch for corrector
    roundPrecision      1e-4;   //1e-16;     //set lower precision for higher stability
    minConcentration    1e-6;  //minimum concentration below which NO ATTEMPT will be made to compute nodes
    computeEigenValues  true;   //if false, this will force the use of the Jacobi-elements as the eigenvalues
    useEqualWeights     false;   //if true, will by-pass eigenvector computation, and use simple estimate of weights

    simpleMeanDiameterId1   2;  //use to specify which moment is used for a simple estimate of the local diameter (numerator)
    simpleMeanDiameterId0   1;  //use to specify which moment is used for a simple estimate of the local diameter (denominator)

};

/*diffusionPhysSpaceProps
{
//    verbose     true;
    useQmom     true;
    gamma0      0.0;
    useLocalDiff  true;
    dynViscosity  1.8e-5;
}*/


```

File: /home/mario/OpenFOAM/mario-2....140603_QMOM/system/controlDict Page 1 of 4

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     cpcFoamCompressible; //compressible solver

minMaxOutputFreq    1000; //frequency to report min/max statistics

//Main Switches for solver
solveFluidFlow;
solveEnergyEqn;     //deactivate in case of stability problems
solveSpeciesEqn;    //deactivate in case of stability problems
solveParticleEqns;    //activate to update particle cloud information

startFrom       startTime; // use "latestTime" to continue simulation from last time

startTime       0;

stopAt          endTime;

endTime         1.2;

deltaT          5e-7;

writeControl    adjustableRunTime;

writeInterval   1e-1;

purgeWrite      15;

writeFormat     ascii;

writePrecision  7;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;

adjustTimeStep  false; //use false in case MOM/QMOM is used!

maxCo           0.6; //decrease in case of stability problems

libs (
     "libOpenFOAM.so"
      "libsimpleSwakFunctionObjects.so"
      "libswakFunctionObjects.so"
      "libfieldFunctionObjects.so"
      "libgroovyBC.so"
     );

```

File: /home/mario/OpenFOAM/mario-2....140603_QMOM/system/controlDict Page 2 of 4

```
functions
{
  //****************************
  //Min/max reporting
  minMaxValues_S
  {
      type fieldMinMax;
      outputControl          timeStep;
      outputInterval $minMaxOutputFreq;
      fields
      (
           S
     );
  }
  minMaxValues_m0
  {
      type fieldMinMax;
      outputControl          timeStep;
      outputInterval $minMaxOutputFreq;
      fields
      (
           m0
     );
  }
  minMaxValues_m1
  {
      type fieldMinMax;
      outputControl          timeStep;
      outputInterval $minMaxOutputFreq;
      fields
      (
           m1
     );
  }
  minMaxValues_m2
  {
      type fieldMinMax;
      outputControl          timeStep;
      outputInterval $minMaxOutputFreq;
      fields
      (
           m2
     );
  }
  minMaxValues_m3
  {
      type fieldMinMax;
      outputControl          timeStep;
      outputInterval $minMaxOutputFreq;
      fields
      (
           m3
     );
  }
  minMaxValues_node0
  {
      type fieldMinMax;
      outputControl          timeStep;
      outputInterval $minMaxOutputFreq;
      fields
      (
           node0
     );
  }
  minMaxValues_node1
  {
      type fieldMinMax;
```

File: /home/mario/OpenFOAM/mario-2....140603_QMOM/system/controlDict Page 3 of 4

```
        outputControl          timeStep;
        outputInterval $minMaxOutputFreq;
        fields
        (
             node1
       );
    }

    //****************************
    //Sampling
    samplesCondenser
    {
      type                        sets;
      interpolationScheme cell; //ATTENTION: AVOID USAGE IF cellPoint or cellPointFace in wedge
  geometry!
      setFormat               raw;
      outputControl          outputTime;

      sets
      (
  /*
       sample_1_x-axis
       {
  //       type    uniform;
          type    midPoint;
          axis    x;
          start   ( 0 0 1e-7 );
          end     ( 0.343 0 1e-7 );
  //        nPoints 20;
        }
       sample_2_eaporator_outlet
        {
  //        type    uniform;
          type    midPoint;
          axis    z;
          start   ( 0.16 0 0.000 );
          end     ( 0.16 0 0.0023 );
  //        nPoints 20;
        }
       sample_3_condensor_inlet
        {
  //        type    uniform;
          type    midPoint;
          axis    z;
          start   ( 0.19 0 0 );
          end     ( 0.19 0 0.0023 );
  //        nPoints 50;
        }
       sample_4_x200
        {
  //        type    uniform;
          type    midPoint;
          axis    z;
          start   ( 0.2 0 0.000 );
          end     ( 0.2 0 0.0023 );
  //        nPoints 20;
        }

       sample_5_axis
        {
  //        type    uniform;
          type    midPoint;
          axis    x;
          start   ( 0 0 1e-7 );
          end     ( 0.26 0 1e-7 );
  //       nPoints 50;
        }
```

```
*/
     sample_6_condensor_outlet
      {
//        type    uniform;
        type    midPoint;
        axis    z;
        start   ( 0.243 0 0 );
        end     ( 0.243 0 0.0023 );
//        nPoints 50;
      }
    );

    fields
    (
            p Y S T m0 m1 m2 m3
    );
  }

  //****************************
  //Probing
  probes
  {
        type            probes;
        functionObjectLibs ("libsampling.so");
        outputControl   outputTime;
        probeLocations
        (
            ( 0.001 0 1e-7 )
            ( 0.200 0 1e-7 )
        );
        fields
        (
            p Y S T m0 m1 m2 m3
       );
  }
};

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

File: /home/mario/OpenFOAM/mario-2....0140603_QMOM/system/fvSolution Page 1 of 2

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    "rho.*"
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       0;
        relTol          0;
    }

    p_rgh
    {
        solver           GAMG;
        tolerance        1e-8;
        relTol           0.01;

        smoother         DIC;

        cacheAgglomeration true;
        nCellsInCoarsestLevel 10;
        agglomerator     faceAreaPair;
        mergeLevels      1;

        maxIter          200;
    }

    p_rghFinal
    {
        $p_rgh;
        relTol          0;
    }

    "(U|T|h|k|epsilon|R)"
    {
        solver          PBiCG;
        preconditioner  DILU;
        tolerance       1e-10; //decrease if unstable, and then increase towards the end of the
simulation
        relTol          0.01;
    }

    "(U|T|h|k|epsilon|R)Final"
    {
        $U;
        relTol          0;
    }

    "(Y)"
    {
        solver          PBiCG;
```

File: /home/mario/OpenFOAM/mario-2....0140603_QMOM/system/fvSolution Page 2 of 2

```
        preconditioner  DILU;
        tolerance       1e-10;
        relTol            1e-3;
    }

    "(Y)Final"
    {
        $U;
        relTol          0;
    }

    "(m)"
    {
        solver          PBiCG;
        preconditioner  diagonal; //diagonal = more stable setting
        tolerance       1e-32;
        relTol          1e-6;   //0;    //
        maxIter         5;
    }

}

PIMPLE
{
    momentumPredictor yes;
    nOuterCorrectors 2; //use minimum 2 if compressible!
    nCorrectors     2;
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}

relaxationFactors
{

    rho             1.0;
    p_rgh           1.0;
    U               1.0;
    h               0.3;  //can be increased to 0.4 to increase rate of convergence
    "(k|epsilon|omega)" 0.3;

}

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

File: /home/mario/OpenFOAM/mario-2....20140603_QMOM/system/fvSchemes Page 1 of 2

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
}

divSchemes
{
    default         none;
    div(phi,U)      bounded Gauss limitedLinearV 1.0; //upwind;
    div(phi,T)      bounded Gauss limitedLinear 1.0; //upwind;  //
    div(phi,Y)      bounded Gauss  upwind;  //limitedLinear 1.0; //
    div(phi,k)      bounded Gauss  limitedLinear 1.0; //upwind;
    div(phi,epsilon) bounded Gauss  limitedLinear 1.0; //upwind;
    div(phi,R)      bounded Gauss  limitedLinear 1.0; //upwind;
    div(R)          bounded Gauss linear;
    div((nuEff*dev(T(grad(U))))) Gauss linear;

    //relevant for compressible solver only
    div((muEff*dev2(T(grad(U))))) Gauss linear;
    div(phi,K)      bounded Gauss  limitedLinear 1.0; //upwind;
    div(phi,h)      bounded Gauss  upwind;  //limitedLinear 1.0; //
    div(phi,m)      bounded Gauss  upwind;      //limitedLinear 1.0; //
}

laplacianSchemes
{
    default         none;
    laplacian(nuEff,U) Gauss linear corrected;
    laplacian(Dp,p_rgh) Gauss linear corrected;
    laplacian(alphaEff,T) Gauss linear corrected;
    laplacian(DkEff,k) Gauss linear corrected;
    laplacian(DepsilonEff,epsilon) Gauss linear corrected;
    laplacian(DREff,R) Gauss linear corrected;
    laplacian(diffEff,Y) Gauss linear corrected;

    //relevant for compressible solver only
    laplacian(muEff,U) Gauss linear corrected;
    laplacian(alphaEff,h) Gauss linear corrected;
    laplacian(Dp,m) Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}
```

File: /home/mario/OpenFOAM/mario-2....20140603_QMOM/system/fvSchemes Page 2 of 2

```

snGradSchemes
{
    default         corrected;
}

fluxRequired
{
    default         no;
    p_rgh;
}


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

File: /home/mario/OpenFOAM/mario-2....stant/thermophysicalProperties Page 1 of 2

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      thermophysicalProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

thermoType
{
    type            heRhoThermo; //General thermophysical model calculation based on enthalpy
    mixture         pureMixture; //General thermophysical model calculation for passive gas mixtures
    transport       const; //sutherland; //model for transport properties, default: const;
    thermo          hConst;     //Constant specific heat cp model with evaluation of enthalpy
    equationOfState rhoConst; // perfectGas
    specie          specie;
    energy          sensibleEnthalpy; //refer to absolute energy where heat of formation is included, and sensible energy where it is not
}

pRef            1.01325e5;
pRefSpecies     pRefSpecies [1 -1 -2 0 0 0 0] $pRef; //Reference pressure for species solver

mixture
{
    specie
    {
        nMoles          1;
        molWeight       28.96; //AIR
    }
    thermodynamics
    {
        Cp              1005; //heat capacity in J/kg/K
        Hf              0;    //heat of fusion
    }

    //Option A: Setting for
    //transport       const;
    //equationOfState rhoConst;
    transport   //for constant transport properties
    {
        mu              1.8915e-05;
        Pr              0.707;
    }
    equationOfState //specify density
    {
        rho 1.1614;
    }

/*
    //Option B: Setting for
    //  transport     sutherland; //model for transport properties, default: const;
    //equationOfState perfectGas
    transport //for Sutherland transport properties, for air, from McQuillan PROPERTIES OF DRY AIR AT ONE ATMOSPHERE
              //for calculation of thermal conductivity see
              //OpenFOAM-2.2.1/src/thermophysicalModels/specie/transport/sutherland/sutherlandTransportI.H
```

File: /home/mario/OpenFOAM/mario-2....stant/thermophysicalProperties Page 2 of 2

```
    {
        As          1.4592e-06;
        Ts          109.1;
    }
*/
}


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