CFD Modelling of Direct Gas Injection Using a Lagrangian Particle Tracking Approach

Mathias Vångö

Thesis for the degree of Master of Science in Engineering
Division of Fluid Mechanics
Department of Energy Sciences
Faculty of Engineering | Lund University
Thesis for the degree of
Master of Science in Engineering, Mechanical Engineering

CFD Modelling of Direct Gas Injection Using a
Lagrangian Particle Tracking Approach

Mathias Vångö
This degree project for the degree of Master of Science in Engineering has been conducted at the Division of Fluid Mechanics, Department of Energy Sciences, Faculty of Engineering, Lund University.

Supervisors at the Division of Fluid Mechanics were Professor Xue-Song Bai and Docent Mehdi Jangi.

Examiner at Lund University was Dr. Robert-Zoltan Szasz.

The project was carried out in close collaboration with MAN Diesel & Turbo in Copenhagen.

Thesis for the Degree of Master of Science in Engineering

ISRN LUTMDN/TMHP-15/5347-SE

ISSN 0282-1990

© 2015 Mathias Vångö samt Energy Sciences

Fluid Mechanics
Department of Energy Sciences
Faculty of Engineering, Lund University
Box 118, 221 00 Lund
Sweden

www.energy.lth.se
Acknowledgements

After five months of work, carried out at the Division of Fluid Mechanics, Faculty of Engineering at Lund University in close collaboration with MAN Diesel & Turbo, I would like to express my gratitude to my main supervisor Prof. Xue-Song Bai for all advices, discussions and feedback throughout the whole process. Further I would like to thank my other supervisor Mehdi Jangi for the advices and discussions regarding implementation strategies and other OpenFOAM related problems. Also I would like to thank Eric Baudoin at MAN Diesel & Turbo for the big interest in this thesis and for all discussions and feedback. I would also like to thank Cheng Gong for help with compilation and other OpenFOAM related questions. Finally I want to thank my father for the countless lunches and lively discussions of not very thesis related topics during the last five months.
Abstract

CFD simulations of direct gas injection, especially in large dual-fuel engines, can be expensive both regarding time and computational power. The nozzle area needs to be resolved with a fine mesh to capture all phenomena and for a full engine model this results in a large amount of cells. A method using a Lagrangian Particle Tracking (LPT) approach was developed to handle gas injection by injecting gaseous parcels into the domain. The gaseous LPT method was implemented by modifying the LPT solver for liquid droplets in dieselFoam, which was already present in OpenFOAM 2.0.x to minimize development efforts. The method was evaluated by comparisons with RANS simulations of fully resolved subsonic jets in a simple chamber geometry, for different cases with varying inlet velocities and initial chamber conditions. It was found that despite that the gaseous LPT method under-predicts the spreading of the jet as compared with the fully resolved approach, resulting in a longer penetration length, the method provides overall reasonable trends regarding velocities and gas mass fraction. Therefore it was found in this thesis that, with a few modifications to the existing dieselFoam solver, it is possible to model direct gas injection of subsonic jet with reasonable results.
# Contents

1 Introduction ................................................. 6
   1.1 Background ........................................... 6
   1.2 Objectives ........................................... 6
   1.3 Scope and limitations ............................... 6

2 Literature study ............................................. 8
   2.1 Mathematical description of turbulent multi-phase flow ............. 8
      2.1.1 Introduction to Lagrangian Particle Tracking .................. 8
      2.1.2 Governing equations .................................. 8
      2.1.3 Boundary conditions .................................. 11
   2.2 Previous research on Gaseous Sphere Injection .................... 11
      2.2.1 Gaseous sphere model of Hessel et al. ....................... 12
      2.2.2 Penetration length definition and dependency ............... 12
   2.3 OpenFOAM CFD package - the Numerical Solver .................. 13
      2.3.1 Introduction to OpenFOAM ............................. 13
      2.3.2 Relevant structure of dieselFoam ......................... 13
      2.3.3 Mathematical operators in OpenFOAM ..................... 14

3 Method ....................................................... 15
   3.1 Method overview ......................................... 15
   3.2 Specification of liquid properties in default dieselSpray library ... 15
      3.2.1 Constant liquid properties ................................ 15
      3.2.2 Temperature dependent liquid properties .................... 16
   3.3 Implementation of the gaseous LPT model in OpenFOAM .......... 18
      3.3.1 Modifications to the liquid properties .................... 18
      3.3.2 Modifications to the evaporation characteristics .......... 22
      3.3.3 Compilation instructions ............................... 23

4 Case setup and key parameters ................................ 24
   4.1 Case setup ............................................... 24
      4.1.1 Geometry and mesh ..................................... 24
      4.1.2 Boundary and initial conditions ......................... 25
      4.1.3 Injector types and models ................................ 25
   4.2 Key parameters ........................................... 26
   4.3 Post-processing .......................................... 27

5 Results and discussion ....................................... 28
   5.1 Sensitivity analysis ...................................... 37

6 Conclusions .................................................. 43
   6.1 Conclusions ............................................. 43
   6.2 Suggestions for future work .............................. 43
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A  Source code for the implemented fuel</td>
<td>46</td>
</tr>
<tr>
<td>B  Source code for the modified thermophysical functions</td>
<td>49</td>
</tr>
<tr>
<td>C  Modifications to the parcel class</td>
<td>57</td>
</tr>
<tr>
<td>D  Evaporation model</td>
<td>58</td>
</tr>
<tr>
<td>E  Compilation instructions</td>
<td>61</td>
</tr>
<tr>
<td>F  Contour plots for Case 2, 3 and 4</td>
<td>63</td>
</tr>
</tbody>
</table>
## List of Figures

2.1 Structure overview of dieselSpray and thermophysicalModels libraries ........ 14
3.1 Illustrative phase diagram .......................................................... 16
3.2 Validation of Sutherland's law ......................................................... 20
3.3 Polynomial of thermal conductivity ............................................... 21
3.4 Schematic representation of the core region .................................... 22
3.5 Gaseous sphere penetration for varying $X_{core}$ .............................. 23
4.1 Wedge shaped geometry ............................................................... 24
4.2 Schematic view of the geometry .................................................... 25
5.1 Contour plots of mass fraction for various time steps with LPT method for Case 1 28
5.2 Contour plots of mass fraction for various time steps with resolved method for Case 1 .......................................................... 29
5.3 Vapour penetration over time for Case 1 ......................................... 30
5.4 Vapour penetration over time for all cases ...................................... 31
5.5 Centerline mass fraction for various time steps .................................. 32
5.6 Centerline velocity for various time steps ....................................... 33
5.7 Centerline gas pressure for various time steps .................................. 34
5.8 Schematic figure of an evaporating sphere ...................................... 34
5.9 Radial mass fraction profiles for various locations downstream .......... 35
5.10 Radial velocity profiles for various locations downstream .................. 36
5.11 Vapour penetration for varying $X_{core}$ ........................................ 38
5.12 Radial profiles at $x/r = 80$ for varying $X_{core}$ ............................ 38
5.13 Vapour penetration for varying grid size ........................................ 39
5.14 Radial profiles at $x/r = 80$ for varying grid size ............................ 39
5.15 Vapour penetration for varying parcel number ............................... 40
5.16 Radial profiles at $x/r = 80$ for varying parcel number .................... 40
5.17 Vapour penetration for varying gaseous sphere size ......................... 41
5.18 Radial profiles at $x/r = 80$ for varying gaseous sphere size ............. 41
5.19 Vapour penetration for varying spray angle ................................... 42
5.20 Radial profiles at $x/r = 80$ for varying spray angle ....................... 42
6.1 Contour plots of mass fraction for various time steps with gaseous LPT method for Case 2 ......................................................... 64
6.2 Contour plots of mass fraction for various time steps with resolved method for Case 2 ......................................................... 64
6.3 Contour plots of mass fraction for various time steps with gaseous LPT method for Case 3 ......................................................... 65
6.4 Contour plots of mass fraction for various time steps with resolved method for Case 3 ......................................................... 65
6.5 Contour plots of mass fraction for various time steps with gaseous LPT method for Case 4 ......................................................... 66
F.6 Contour plots of mass fraction for various time steps with resolved method for Case 4.
List of Tables

2.1 Mathematical operators in OpenFOAM ........................................ 14
3.1 Constant liquid properties ..................................................... 15
3.2 Temperature dependent liquid properties ............................... 17
3.3 Summary of thermophysical functions ................................... 17
3.4 Implemented constant liquid properties ................................. 18
3.5 Reference viscosity ............................................................. 19
3.6 Coefficients for thermal conductivity polynomial .................... 21
4.1 Geometry dimensions ............................................................ 25
4.2 Boundary and initial conditions for the cases ......................... 25
Chapter 1

Introduction

1.1 Background

The MAN Diesel & Turbo two-stroke GI engines can be operated on natural gas, which is injected directly into the cylinder at high pressure. An engine that can be run with multiple fuels is commonly referred to as a dual-fuel engine. This dual fuel configuration is mostly used for engines operating at a high load at relatively constant speed which makes it suitable for large ships. Natural gas is a good fuel option because compared to heavy fuel oil the exhaust is significantly cleaner. The emission of $\text{SO}_x$, $\text{NO}_x$ and $\text{CO}_2$ are greatly reduced which is helpful to fulfil the emission regulations for ships, especially when operating in Emission Control Areas (ECA), which is one reason why the dual fuel configuration is desirable. Another reason is the flexibility. The ratio between diesel and gas can easily be optimized depending on the diesel/gas prices and availability. However, natural gas is difficult to combust hence some assistance is needed to ignite. This can be done either by a spark-plug or as in MAN Diesel & Turbo’s case where a small portion of diesel is injected, a so called pilot fuel. This means that the engine can’t operate at 100% gas, a small portion of diesel is always needed, the operating area is roughly from 0% to about 97% natural gas. With this configuration, any mix of the fuels within this range is possible to optimize for the fuel cost and availability [1–3].

The gas injection could be simulated with either Reynolds Averaged Navier-Stokes (RANS) or Large Eddy Simulation (LES) turbulence modelling but because of the small gas injection area and relatively large cylinder volume it requires huge amount of cells to capture all phenomena close to the nozzle. Thus those simulations are expensive regarding both time and computational power, even more so if a full model of the engine is considered. Thus an alternative approach is desired to make these simulations cheaper and more manageable.

1.2 Objectives

The aim of this thesis is to develop and implement a model for direct gas injection using a Lagrangian Particle Tracking (LPT) approach in the open source CFD package OpenFOAM 2.0.x and evaluate this approach. The LPT approach is generally intended for a liquid fuel spray but it could be used to model direct gas injection with some care regarding the fuel properties and evaporation [4,5]. The existing LPT solver dieselFoam that is already present in OpenFOAM will be modified in order to minimize development efforts.

1.3 Scope and limitations

The scope of this thesis is to develop and evaluate the LPT approach for gas injection, no in-depth study of the physics of the gas jet will be performed. Additionally, to be able to
perform this thesis in a reasonable amount of time and reduce the possible sources of errors a few limitations had to be made. They are listed below:

- Simulations were carried out on a generic, constant volume chamber geometry.
- The geometry was 2D axisymmetric.
- Only correctly expanded jets were looked at, i.e. only subsonic inlet velocities.
- The model was not validated against any real experiment but against fully resolved simulations.
- The simulation was limited to the RANS framework, although the method can be applicable in LES.
Chapter 2

Literature study

2.1 Mathematical description of turbulent multi-phase flow

2.1.1 Introduction to Lagrangian Particle Tracking

Lagrangian Particle Tracking is a method to describe multi-phase flows by introducing particles to the gas phase and let them interact. The particles are commonly known as a dispersed phase and the fluid particles are interacting with the continuous phase. The particle motion is described by the Newtonian motion equations and it interacts with the continuous phase with source terms introduced in the governing equations of the continuous phase. The dispersed phase is solved within a Lagrangian coordinate system and the continuous phase within an Eulerian coordinate system [6]. The difference is that a Eulerian coordinate system is fixed in space and a Lagrangian is not. To get a better understanding an analogy could be used where the Eulerian system can be thought of as a system of fixed weather stations that record data and the Lagrangian system as a weather balloon that follows the wind and record data [7]. Commonly, the term parcel is introduced. It describes a certain number of particles that are assumed to have the same properties and follow the same path. This method is used because it is not practical to track all particles regarding time and computational power.

2.1.2 Governing equations

Gas phase equations

Because RANS turbulence modelling will be used the gas phase is described with the averaged Navier-Stokes partial differential equations. The flow can not be assumed to be incompressible hence it is described by the Favre averaged Navier-Stokes equations. Favre averaged means that the equations are density- and time averaged and the procedure is shown in Eqs. (2.1) - (2.4) [8].

Time averaging for any dependant variable, $\Phi$.

$$\overline{\Phi} = \frac{1}{T} \int_{T} \Phi(t) dt$$ (2.1)

$$\Phi = \overline{\Phi} + \Phi'$$ (2.2)

Density weighted time averaging of $\Phi$.

$$\hat{\Phi} = \frac{\rho \Phi}{\rho}$$ (2.3)

$$\Phi = \hat{\Phi} + \Phi''$$ (2.4)

The Favre-averaged Navier-Stokes are given in Eqs. (2.5), (2.6), (2.8) and (2.9).
Continuity:

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_j}(\rho \tilde{u}_j) = S^s_{\rho}$$ \quad (2.5)

Momentum:

$$\frac{\partial}{\partial t} (\rho \tilde{u}_i) + \frac{\partial}{\partial x_j}(\rho \tilde{u}_i \tilde{u}_j) = \frac{\partial}{\partial x_j} \left( -\rho \delta_{ij} + \tau_{ji} - \rho u_i^I u_j^I \right) + S^s_{u,i}$$ \quad (2.6)

where $\tau_{ji}$ is the viscous stress tensor defined as

$$\tau_{ji} = \mu \left( \frac{\partial \tilde{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_j}{\partial x_i} - \frac{2}{3} \frac{\partial \tilde{u}_k}{\partial x_k} \delta_{ji} \right)$$ \quad (2.7)

and $-\rho u_i^I u_j^I$ is the Reynolds-stress tensor.

Species transport:

$$\frac{\partial}{\partial t} (\rho \tilde{Y}_i) + \frac{\partial}{\partial x_j}(\rho \tilde{Y}_i \tilde{u}_j) = \frac{\partial}{\partial x_j} \left( \rho D \frac{\partial \tilde{Y}_i}{\partial x_j} - \rho u_j^I \tilde{Y}_i^I \right) + S^s_{\rho,i}$$ \quad (2.8)

$D$ is the mass diffusion coefficient.

Energy:

$$\frac{\partial}{\partial t} (\rho \tilde{E}) + \frac{\partial}{\partial x_j}(\rho \tilde{E} \tilde{u}_j) = \frac{\partial}{\partial x_j} \left( -p \tilde{u}_j + u_i \tau_{ji} - q_j - \rho u_j^I \tilde{E}^I \right) + S^s_{E}$$ \quad (2.9)

$\tilde{E}$ is the total energy defined as

$$\tilde{E} = \tilde{e} + \frac{\tilde{u}_i \tilde{u}_i}{2}$$ \quad (2.10)

and $q_j$ is the heat flux defined as below.

$$q_j = -\lambda \frac{\partial T}{\partial x_j}$$ \quad (2.11)

$\lambda$ is the thermal conductivity and $\tilde{e}$ is the internal energy.

The terms $S^s_{\rho}, S^s_{\rho,i}, S^s_{u,i}$ and $S^s_{E}$ are source terms due to the parcels and are obtained by studying the Lagrangian representation of the liquid phase.

**Source terms from the liquid phase**

$S^s_{\rho}$ is the gaseous sphere evaporation rate of all spray parcels in the cell as shown below

$$S^s_{\rho} = -\frac{1}{V_{cell}} \sum N_p \frac{dm_{gs}}{dt}$$ \quad (2.12)

where $m_{gs}$ is the mass of a single gaseous sphere and $N_p$ is the number of gaseous spheres in the parcel. The time derivative of $m_{gs}$ is the evaporation rate of a gaseous sphere and is given by the relationship in Eq.(2.13),

$$\frac{dm_{gs}}{dt} = m_{gs} = -\frac{m_{gs}}{\tau_e}$$ \quad (2.13)

where $\tau_e$ is an evaporation relaxation time [9].

**Momentum:**

Similarly to $S^s_{\rho}$, the rate of momentum change, $S^s_{u,i}$ can be expressed as:

$$S^s_{u,i} = -\frac{1}{V_{cell}} \sum N_p m_{gs} \frac{du_{gs,i}}{dt}$$ \quad (2.14)
The acceleration of a gaseous sphere can be written as
\[
\frac{du_{gs,i}}{dt} = -u_{gs,i} - u_{i} \tau_{u,i} = \frac{u_{rel,i}}{\tau_{u,i}}
\]  
(2.15)

where \( \tau_{u,i} \) is a momentum relaxation time that is a function of droplet size, velocity, density and drag coefficient. This implies that the only forces working on the gaseous sphere is the drag force and the drag coefficient \( C_D \) can be calculated as \([9,10]\),
\[
C_D = \begin{cases} 
\frac{24}{\pi Re_{gs}} \left(1 + \frac{1}{5} Re_{gs}^{2/3}\right) & \text{for } Re_{gs} \leq 1000 \\
0.426 & \text{for } Re_{gs} > 1000 
\end{cases}
\]  
(2.16)

where \( Re_{gs} \) is the Reynolds number for a gaseous sphere defined as
\[
Re_{gs} = \frac{\rho |u_{rel,i}| d_{gs}}{\mu}
\]  
(2.17)

Species:
The same as \( S^s_{\rho, i} \) can be expressed as
\[
S^s_{\rho,i} = -\frac{1}{V_{cell}} \sum N_p \frac{dm_{gs,i}}{dt}
\]  
(2.18)

for species \( i \).

Energy:
\( S^s_E \) is the source term that accounts for exchange of energy between the phases and it can be expressed as
\[
S^s_E = -\frac{1}{V_{cell}} \sum N_p \frac{m_{gs}}{dt}
\]  
(2.19)

The droplet temperature change is given by the following equation \([9]\),
\[
\frac{dT_{gs}}{dt} = T - T_{gs} \cdot \frac{1}{\tau_h} \cdot \frac{h_v(T_{gs})}{c_{l,gs}} \frac{1}{\tau_e}
\]  
(2.20)

where \( c_{l,gs} \) is the liquid specific heat, \( \tau_h \) is a characteristic heat transfer relaxation time and \( \tau_e \) is as already mentioned the evaporation relaxation time.

Turbulence
The standard \( k - \epsilon \) turbulence model is used to model the turbulence. In this model, the Boussineq eddy viscosity assumption is used to approximate Reynolds-stress tensor as
\[
-\rho u_i u_j = \mu_t \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) - \frac{2}{3} \rho k \delta_{ij}
\]  
(2.21)

where \( \mu_t \) is the turbulent viscosity defined as
\[
\mu_t = \bar{\rho} C_{\mu} \frac{k^2}{\epsilon}
\]  
(2.22)

and \( C_{\mu} \) is a model constant.

The equations (2.23) and (2.24) are standard the \( k - \epsilon \) equations for compressible flows.
\[
\frac{\partial}{\partial t} (\bar{n} k) + \frac{\partial}{\partial x_i} (\bar{n} k \bar{u}_i) = \frac{\partial}{\partial x_j} \left[ \left( \frac{\mu + \mu_t}{\sigma_k} \right) \frac{\partial k}{\partial x_j} \right] + P_k - \bar{n} + \dot{W}^s
\]  
(2.23)
\[
\frac{\partial}{\partial t} (p\epsilon) + \frac{\partial}{\partial x_i} (p\epsilon \tilde{u}_i) = \frac{\partial}{\partial x_j} \left[ \left( \mu + \frac{\mu_t}{\sigma_t} \right) \frac{\partial \epsilon}{\partial x_j} \right] + C_{e1} \frac{\epsilon}{k} - C_{e2} p \frac{\epsilon^2}{k} + C_s \dot{W}_s \tag{2.24}
\]

\(C_{e1}, C_{e2}, \sigma_k, \sigma_t,\) and \(C_s\) are also constants. \(\dot{W}_s\) is the spray-induced turbulence source term which describes the rate of change of turbulent kinetic energy. It can be shown that \(\dot{W}_s < 0\) and thus this term always depletes turbulent kinetic energy \([11]\).

\(\dot{W}_s\) is defined below as

\[
\dot{W}_s = \frac{1}{V_{cell}} \sum N_p m_{gs} \frac{du_{gs,i}}{dt} u_{i}^{''} \tag{2.25}
\]

### 2.1.3 Boundary conditions

To close out the system of equations, boundary conditions for all partial differential equations have to be specified. All boundaries of the domain needs to be fully specified to solve the equations. Boundary conditions can be set in different ways for the walls, inlet and outlet. In this case no outlet is present in the chamber thus only boundary conditions for inlet and walls are necessary. Boundary conditions can either be specified as a Dirichlet or Neumann boundary condition, or in other words by specifying a fixed value, or the gradient normal to the boundary. A commonly used boundary conditions for walls is a no-slip condition. No-slip means that all velocities at the wall is zero, which is the case for viscous flows. In RANS simulations, since the wall boundary layer is not resolved, wall-functions are used as wall boundary conditions \([12]\).

### 2.2 Previous research on Gaseous Sphere Injection

Some research has been done on the topic gaseous sphere injection, primarily by Hessel, Abani, Aceves and Flowers in the papers "Gaseous Fuel Injection Modeling using a Gaseous Sphere Injection Methodology" \([4]\) and "Application of gaseous sphere injection method for modeling under-expanded H\(_2\) injection" \([5]\). Gaseous sphere is the gaseous analogy, equivalent for a liquid droplet. In their research a few modifications were made to the KIVA3V CFD code, which has capabilities of handling liquid fuel spray injection, to handle gas injection.

Oulette found when modelling gas jets, that "the momentum injection rate must be reproduced if one wants to reproduce the mixing rate". Hessel et al. \([4]\) then stated that it should not matter how the fuel enters the domain as long as it enters with the same momentum injection rate. The momentum injection rate is defined as in Eq. (2.26) where \(\dot{m}\) is the inlet mass flow, \(U_{noz}\) is the exit velocity from the nozzle, \(\rho_{noz}\) is the nozzle exit density and \(A_{noz}\) the nozzle exit area.

\[
\dot{M} = \dot{m} \cdot U_{noz} = \rho_{noz} A_{noz} U_{noz}^2 \tag{2.26}
\]

Oulette also found out that "for sprays with droplets of 5 and 10 microns (Sauter mean radius) injected in a chamber at 1200 K, the mixing rates of gaseous jets and sprays were much the same for equivalent nozzle momentum and mass injection rate. Furthermore, when the cone angle of the sprays and spreading angle of the gaseous jet were roughly equal, the penetration was also similar". Hence if the gaseous spheres are injected at a roughly equal cone angle to the gas jet spreading angle, the penetration length should be reproduced \([4, p. 2]\). Here, Sauter mean radius is an average of particle size.

One of the benefits of using a gaseous sphere injection approach is that re-meshing is not necessary if a different injector or injector position is chosen because no gas inlet patch is required on the domain boundary. It also means that the injector does not necessary have to be located on the boundary which creates more freedom and flexibility. Another advantage is as stated before, that a coarser mesh can be used since the mesh does not need to be resolved close to the injector, which in the end lead to shorter and cheaper simulations \([4]\).
Hessel et al. [4] did several tests on simple gas injection experiments to evaluate and validate the method. Cases of air injected into air, methane into air and helium into air, ranging from sub-sonic to super-sonic jets were simulated and evaluated with experiments. Overall, the studies showed that the model seems to capture the general trends of the experiments but in some cases the jet penetration was under-predicted, which was suggested that it could be due to an over-estimation of diffusion in radial direction. However, the helium into air case showed a slight over-prediction at late stages of the injection. It was concluded that the deviation generally occur at times after the fuel transition to the gaseous phase which was a positive sign because this issue have to be overcome with any chosen method [4].

2.2.1 Gaseous sphere model of Hessel et al.

According to Hessel et al. [4] there are five requirements to get gaseous fuel into a domain. They are given below:

1. A fuel inlet must be located in the computational domain with proper orientation.
2. Fuel must have the correct properties when it enters the computational domain.
3. The jet flow characteristics must be modelled.
4. The injected parcels and the ambient gas must interact.
5. At some point the injected parcels must transition from distinct entities to being part of the ambient gas.

Fuel properties in terms of temperature, density and velocity must have the correct properties when it enters the domain. For simple cases this can be calculated with 1D gas dynamics from the upstream conditions. Entrainment is created when the particles interact with the ambient gas due to the exchange of momentum. Momentum is exchanged because the gaseous spheres experience drag forces from the ambient when injected. Thus, every property that affects the drag need to be considered in the model. Mass and energy are exchanged when the gaseous sphere "evaporate". This is technically not an evaporation since the sphere is not a liquid droplet, it is however a transition where the mass and energy of the gaseous sphere cease to exist and becomes a part of the ambient gas. This transition will still be referred to as evaporation throughout this thesis.

In Hessel et al. [4] model the evaporation occurs when a gaseous sphere travels past the jet core length. It was suggested an inviscid core length defined as in Eq. (2.27) along the jet center axis. It should be noted that with this methodology another parameter is introduced in the model and $X_{core}$ may vary from case to case or injector to injector. The evaporation occurs when a particle is outside a cylindrical region with a length of $X_{core}$ and a radius of $r_{noz}$ from the injector position.

$$X_{core} = 12.5 \cdot r_{noz}$$ (2.27)

Changes were also made to modify the Re-Normalisation Group (RNG) $k-\epsilon$ turbulence model in terms of turbulent kinetic energy and turbulence length scale at different locations. This was done because the RNG $k-\epsilon$ model is known to over-predict jet diffusion and thus under-predict jet penetration.

2.2.2 Penetration length definition and dependency

Oullette and Hill [13] states that the penetration of a gas jet depends on the square root of injection time. The penetration length is strictly dependent on the momentum injection rate
and not injection pressure, velocity or nozzle diameter. Eq. (2.28) was established where \( z_t \) is the jet penetration length, \( \dot{M}_n \) the momentum injection rate, \( \rho_a \) the chamber density, \( t \) the injection time and \( \Gamma \) a constant.

\[
\frac{z_t}{\left( \frac{\dot{M}_n}{\rho_a} \right)^{1/4} t^{1/2}} = \Gamma
\]

(2.28)

Oullette and Hill [13] shows, based on several experiments with a wide range of injection pressures and velocities, that \( \Gamma = 3.0 \pm 0.1 \). However, Eq. (2.28) is only valid for distances greater than 20 nozzle diameters downstream, free jets with a minimum Reynolds number of 30 000 and for times shorter than the injection duration. Thus it is only valid when there is continuous feeding of the jet.

When measuring the penetration length certain definition of the penetration is needed. There is no clear definition and the penetration length can be defined either with the velocity or mass fraction. In Oulette and Hill [13] an attempt to a definition was made by taking a point roughly halfway between the jet forefront and the beginning of the jet head. This corresponded, for distances greater than 20 nozzle diameters downstream to a mass fraction 3%. Thus, the mass fraction of fuel of 3% along the jet axis will be used to define jet penetration throughout this thesis.

2.3 OpenFOAM CFD package - the Numerical Solver

2.3.1 Introduction to OpenFOAM

The OpenFOAM CFD Toolbox is a free, open source CFD package. OpenFOAM stands for Open Field Operation and Manipulation and it is widely used in many fields of engineering and science. OpenFOAM includes solvers to solve all different kinds of complex flows involving e.g. turbulence and chemical reactions and different utilities for pre- and post-processing. OpenFOAM is a desired CFD package because it is open source and it offers complete freedom of customization with great flexibility. OpenFOAM is written in the programming language C++ and all executable applications are linked to a collection of libraries. The solvers which will be used in this thesis are dieselFoam for the gaseous LPT method and reactingFoam for the fully resolved method [14].

2.3.2 Relevant structure of dieselFoam

In this section an overview of the structure of relevant parts in OpenFOAM to this thesis is presented. DieselFoam was the starting point of the structure investigation as this solver was the baseline. The main libraries relevant to this work were thermophysicalModels and dieselSpray for liquid properties and for the spray/parcels respectively. Fig. 2.1 shows a hierarchical structure of the thermophysicalModels and dieselSpray libraries. In the dieselSpray library, classes for calculations of each individual parcel are found. How the interaction, predicted movement and update of the properties are handled are all described in the parcel class. The spray class describes the spray as a whole, interpreted as a cloud of a certain number of parcels. In the injector class several pre-defined injector types can be found. It includes models for e.g. common rail injector, unit injector and defined injector. Models for describing evaporation, breakup, drag, injection, heat transfer, etc., are defined in spraySubModels.

In the thermophysicalModels library different models for describing the properties of the flow are found. Mostly relevant for this thesis is the liquid properties and thermophysical properties. In liquid properties, classes for several different pre-defined fuels are located, containing
information of fluid dependent properties and coefficients for the temperature dependent thermophysical functions. The thermophysical functions are located in thermophysicalFunctions and are functions used to calculate temperature dependent fuel properties such as density, specific heat capacity and enthalpy.

2.3.3 Mathematical operators in OpenFOAM

OpenFOAM uses the basic classes scalar, vector and tensor for calculations with tensors of respective order 0, 1 and 2 [15]. To better understand the code presented in this thesis, some mathematical operators for these classes are listed in Table 2.1. A complete list of all mathematical operators can be found in [15, p. 22-23].

<table>
<thead>
<tr>
<th>Mathematical Description</th>
<th>Description in OpenFOAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>a + b</td>
<td>a + b</td>
</tr>
<tr>
<td>a - b</td>
<td>a - b</td>
</tr>
<tr>
<td>sa</td>
<td>s * a</td>
</tr>
<tr>
<td>a/s</td>
<td>a / s</td>
</tr>
<tr>
<td>a^2</td>
<td>sqr(a)</td>
</tr>
<tr>
<td>a^n</td>
<td>pow(a, n)</td>
</tr>
<tr>
<td>ln s</td>
<td>log(s)</td>
</tr>
<tr>
<td>sqrt s</td>
<td>sqrt(s)</td>
</tr>
<tr>
<td>e^s</td>
<td>exp(s)</td>
</tr>
</tbody>
</table>

Table 2.1: Mathematical operators in OpenFOAM
Chapter 3

Method

3.1 Method overview

A model was created and implemented to make the liquid fuel behave as a gas and a new evaporation model was created to obtain the desired evaporation characteristics found in the literature study. After the implementation a simple mesh was created with the blockMesh utility (which is a standard utility in OpenFOAM). A few cases were set up with varying inlet- and chamber conditions to evaluate the gaseous LPT method. The cases were run with both fully resolved gas injection and the newly implemented gaseous LPT model. Additionally a sensitivity analysis was performed for new spray relevant parameters introduced. Post-processing was done in Matlab and Paraview. Detailed information about the thesis work process is described in this chapter.

3.2 Specification of liquid properties in default dieselSpray library

3.2.1 Constant liquid properties

The liquids, in the liquid library, are initialized with the constant, fluid dependent properties shown in Table 3.1 and coefficients for the temperature dependent properties shown in Table 3.2.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>molecular weight</td>
</tr>
<tr>
<td>$T_c$</td>
<td>critical temperature</td>
</tr>
<tr>
<td>$p_c$</td>
<td>critical pressure</td>
</tr>
<tr>
<td>$V_c$</td>
<td>critical volume</td>
</tr>
<tr>
<td>$Z_c$</td>
<td>critical compressibility factor</td>
</tr>
<tr>
<td>$T_t$</td>
<td>triple point temperature</td>
</tr>
<tr>
<td>$p_t$</td>
<td>triple point pressure</td>
</tr>
<tr>
<td>$T_b$</td>
<td>normal boiling temperature</td>
</tr>
<tr>
<td>dipm</td>
<td>dipole moment</td>
</tr>
<tr>
<td>$\omega$</td>
<td>Pitzer’s acentric factor</td>
</tr>
<tr>
<td>$\delta$</td>
<td>solubility parameter</td>
</tr>
</tbody>
</table>

Table 3.1: Constant liquid properties

The critical point is the end of the phase curve in a phase diagram. Above the critical point the difference between liquid- and gas phase disappears and the fluid is said to be supercritical. The critical temperature is the corresponding temperature to the critical point and can also be
defined as the highest temperature where the fluid can exist in liquid state. With the same reasoning the critical pressure is the corresponding pressure to the critical point and it is the highest pressure where the fluid can be in the gaseous phase. The critical volume is the molar volume at the critical point and the critical compressibility factor is the compressibility defined as in Eq. (3.1) at the critical point. The compressibility factor is used to correct the ideal gas law to behave more like a real gas, thus for an ideal gas it is equal to unity [16, 17].

\[ Z = \frac{pV}{RT} \tag{3.1} \]

The triple point is defined as the point where all phases (solid, liquid and gas) coexists. The critical- and triple point are shown in the illustrative phase diagram, Fig. 3.1.

\[ \text{Figure 3.1: Illustrative phase diagram, [18]} \]

The normal boiling temperature is the boiling temperature of the fluid at atmospheric pressure and dipole moment is a measure of forces between molecules due to positive and negative charges. Pitzer’s acentric factor is a measurement of non-sphericity of molecules and fluids with completely spherical molecules have a \( \omega \) value of zero. The solubility parameter gives an indication of how well a fluid dissolves in another fluid [19–21].

### 3.2.2 Temperature dependent liquid properties

The temperature dependent liquid properties are calculated with thermophysical functions. The properties and respectively thermophysical functions are listed in Table 3.2. Coefficients for these functions are supplied for each fuel from the liquid properties library. As seen in Section 2.3.2 the thermophysical functions library consists of API- and NSRDS functions, API stands for Americal Petroleum Institute and NSRDS for National Standard Reference Data System.

Vapour pressure is the pressure of a vapour in a closed container in thermodynamic equilibrium with its liquid phase. The vapour pressure is strictly temperature dependent and when the vapour pressure is equal to the external pressure, it is boiling. The latent heat of vaporization is the amount of energy needed to vaporize a liquid at constant temperature and the second virial coefficient is a temperature dependent coefficient used in the virial equation of state for gases. Thermal conductivity is the ability of a fluid to conduct heat and it is the opposite of thermal resistance, thus a high value means that the liquid is good at transporting heat. [22–24]
<table>
<thead>
<tr>
<th>Property</th>
<th>Thermophysical function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho$</td>
<td>density</td>
</tr>
<tr>
<td>$p_v$</td>
<td>vapour pressure</td>
</tr>
<tr>
<td>$h_l$</td>
<td>latent heat of vaporization</td>
</tr>
<tr>
<td>$c_p$</td>
<td>specific heat capacity</td>
</tr>
<tr>
<td>$h$</td>
<td>enthalpy</td>
</tr>
<tr>
<td>$\mu_g$</td>
<td>dynamic viscosity</td>
</tr>
<tr>
<td>$c_{pg}$</td>
<td>specific heat capacity, gas phase</td>
</tr>
<tr>
<td>$B$</td>
<td>second virial coefficient</td>
</tr>
<tr>
<td>$\mu$</td>
<td>dynamic viscosity</td>
</tr>
<tr>
<td>$k_g$</td>
<td>thermal conductivity</td>
</tr>
<tr>
<td>$k$</td>
<td>thermal conductivity, gas phase</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>surface tension</td>
</tr>
<tr>
<td>$D$</td>
<td>vapour diffusivity coefficient</td>
</tr>
</tbody>
</table>

Table 3.2: Temperature dependent liquid properties

NSRDSfunctions

Below in Table 3.3 a summary of the thermophysical functions and what they do in OpenFOAM is presented. The letters a-f are the coefficients supplied for each liquid fuel.

<table>
<thead>
<tr>
<th>Thermophysical function</th>
<th>Returned value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSRDSfunc0</td>
<td>$a + b \cdot T + c \cdot T^2 + d \cdot T^3 + e \cdot T^4 + f \cdot T^5$</td>
</tr>
<tr>
<td>NSRDSfunc1</td>
<td>$e^{a+\frac{b}{T}+c\ln(T)+dT}$</td>
</tr>
<tr>
<td>NSRDSfunc2</td>
<td>$\frac{a \cdot T^b}{1+c/T+d/T^e}$</td>
</tr>
<tr>
<td>NSRDSfunc4</td>
<td>$a + \frac{b}{T} + \frac{c}{T^4} + \frac{d}{T^7} + \frac{e}{T^9}$</td>
</tr>
<tr>
<td>NSRDSfunc5</td>
<td>$a + b \cdot T + c \cdot T^2 + d \cdot T^3 + e \cdot T^4 + f \cdot T^5$</td>
</tr>
<tr>
<td>NSRDSfunc6</td>
<td>$a \cdot (1 - T_r)^{b+c T_r+d T_r^2+e T_r^3}$ where $T_r$ is defined as $T_r = T/T_c$</td>
</tr>
<tr>
<td>NSRDSfunc7</td>
<td>$a + b \left( \frac{c/T}{\sinh(c/T)} \right)^2 + d \left( \frac{e/T}{\cosh(e/T)} \right)^2$</td>
</tr>
<tr>
<td>NSRDSfunc14</td>
<td>$\frac{a^2}{T} - b + 2ac \cdot t + ad \cdot t^2 + \frac{c^2}{3} \cdot t^3 + \frac{cd}{2} \cdot t^4 + \frac{d^2}{5} \cdot t^5$ where $t$ is defined as $t = 1 - T/T_c$</td>
</tr>
<tr>
<td>APIfunc</td>
<td>$3.6059 E^{-3} \cdot (1.8 T)^{1.75} \left( \frac{\alpha}{\beta} \right)$ where $\alpha$ and $\beta$ are helper functions defined as $\alpha = \sqrt{\frac{1}{w_f} + \frac{1}{w_a}}$, $\beta = \left( \sqrt{\alpha} + \sqrt{\beta} \right)^2$</td>
</tr>
</tbody>
</table>

Table 3.3: Summary of thermophysical functions
3.3 Implementation of the gaseous LPT model in OpenFOAM

As seen previously in Section 2.2, the liquid properties have to be modified to act as a gas. The evaporation needs to be controlled in a way to resemble an inviscid core, which introduces a new parameter, $X_{\text{core}}$. $X_{\text{core}}$ refers to the distance downstream along the x-axis where the gaseous spheres evaporate.

3.3.1 Modifications to the liquid properties

All properties which influence the momentum transfer between the sphere and gas phase have to be modified. As seen previously in Section 2.1.2, the properties that affects the momentum transfer are the density, gaseous sphere size, velocity and dynamic viscosity. A description of the implementation of the liquid properties is presented in this section, relevant source code can be found in Appendix A and Appendix B.

Constant properties

A new fuel, myCH4 was created with the same properties as the already present fuel $N_2$ and added to the liquid fuel library. The fluid dependent constants from Table 3.1 had to be modified and set to appropriate values to match methane in gaseous state. The molecular weight for methane was taken from NIST Chemistry WebBook, [25] and was found to be $W = 16.0425 \text{ kg/kmol}$. Since in this approach only the gas phase is present even though the parcel is treated as a liquid, the values for critical- and triple-point are not of importance. The critical conditions were assigned a large value which the fluid will never reach and the triple-point was set to a small value for the same reason. The dipole moment, Pitzer’s acentric factor and the solubility parameter were all left unchanged because it was assumed to not have any significant impact on the result. The values are summarised in Table 3.4 where GREAT and SMALL are constants defined in the scalar class in OpenFOAM. GREAT is a large value and similarly SMALL is a small value. SMALL is often used instead of zero to avoid any illegal mathematical operations being performed. The exact values vary depending on how OpenFOAM is compiled.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W$</td>
<td>16.0425</td>
</tr>
<tr>
<td>$T_c$</td>
<td>GREAT</td>
</tr>
<tr>
<td>$p_c$</td>
<td>GREAT</td>
</tr>
<tr>
<td>$V_c$</td>
<td>GREAT</td>
</tr>
<tr>
<td>$Z_c$</td>
<td>GREAT</td>
</tr>
<tr>
<td>$T_i$</td>
<td>SMALL</td>
</tr>
<tr>
<td>$p_t$</td>
<td>SMALL</td>
</tr>
<tr>
<td>$T_b$</td>
<td>GREAT</td>
</tr>
<tr>
<td>dipm</td>
<td>0.0</td>
</tr>
<tr>
<td>$\omega$</td>
<td>0.0403</td>
</tr>
<tr>
<td>$\delta$</td>
<td>$9.0819 e^3$</td>
</tr>
</tbody>
</table>

Table 3.4: Implemented constant liquid properties
Temperature dependent properties

The temperature dependent properties that affect the drag are density and dynamic viscosity. The changes to those properties are presented in this section. However, all properties in Table 3.2 were modified for completeness of the model and are also presented.

Density

The density was originally calculated with a NSRDS function. To get a density representative of a gas, the thermophysical function was modified to calculate the density with the ideal gas law, Eq. (3.2). \( R \) is the specific gas constant for the fuel.

\[
\rho = \frac{p}{RT} \tag{3.2}
\]

Dynamic viscosity

The dynamic viscosity was modified to be calculated with Sutherland’s law, Eq. (3.3).

\[
\mu = \mu_{ref} \left( \frac{T}{T_{ref}} \right)^{3/2} \frac{T_{ref} + S}{T + S} \tag{3.3}
\]

\( T_{ref} \) is a reference temperature, \( \mu_{ref} \) is the corresponding dynamic viscosity at reference temperature and \( S \) is the Sutherland temperature. The Sutherland temperature is a fluid dependent constant which can either be obtained from a table or calculated with two reference points. If a new constant \( C_1 \) is introduced as,

\[
C_1 = \frac{\mu_{ref}}{T_{ref}^{3/2}} (T_{ref} + S) \tag{3.4}
\]

Eq. (3.3) can be written as:

\[
\mu = C_1 \frac{T^{3/2}}{T + S} \tag{3.5}
\]

Eq. (3.5) was the implemented equation for the dynamic viscosity. In order to acquire the constants for methane, two reference points were chosen. Reference temperatures and corresponding viscosities were taken from a NIST table [26] and are shown in Table 3.5.

<table>
<thead>
<tr>
<th>( T_{ref} ) (K)</th>
<th>( \mu_{ref} ) (( \mu \text{Pa} \cdot \text{s} ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
<td>4.36</td>
</tr>
<tr>
<td>1000</td>
<td>27.65</td>
</tr>
</tbody>
</table>

Table 3.5: Reference viscosity

By solving Eq. (3.5) for the two reference points, values for the constants were acquired and found to be \( S = 1.5789 \cdot 10^2 \text{K} \) and \( C_1 = 1.0124 \cdot 10^{-6} \text{ Pa} \cdot \text{s} / \sqrt{\text{K}} \).

To ensure the validity of Sutherland’s law, data from the NIST table and calculation with Eq. (3.3) were plotted for the whole temperature range and are shown in Fig. 3.2. It can be seen that the correlation between dynamic viscosity and temperature is well described.

Vapour pressure

As seen in Table 3.2, both the vapour pressure and dynamic viscosity are calculated with the same thermophysical function, NSRDSfunc1. This is not a desired behaviour, hence a new class was created and named NSRDSfunc_pv. The class NSRDSfunc_pv was modified to instead of
calculating the vapour pressure with a thermophysical function it is treated as a constant value. The vapour pressure is used to determine if the liquid is boiling or not. Boiling is not a condition present for a gaseous sphere thus vapour pressure is set to zero to avoid reaching boiling state in the code.

**Latent heat of vaporization**

As no evaporation actually occurs, no heat is needed to vaporize the fuel. Thus the latent heat of vaporization is set to a constant value of zero.

**Specific heat capacity**

The calculation of specific heat capacity was changed to be calculated with the NASA polynomial shown in Eq. (3.6) [27, p. 9]. The specific heat capacity is already calculated this way for the gas phase and the method was borrowed to calculate the specific heat capacity for the parcels as well. As it is already present in OpenFOAM, the coefficients $a_1 - a_5$ in Eq. (3.6) are already defined for different species. The coefficients are located in a table called CHEMKIN. In this format a total of 14 coefficients are specified, seven for high temperatures and seven for low. A temperature limit to determine if it should be treated as a low or high temperature is also specified.

\[
\frac{C_p}{R} = a_1 + a_2 \cdot T + a_3 \cdot T^2 + a_4 \cdot T^3 + a_5 \cdot T^4
\]  
(3.6)

**Enthalpy**

Similar to the specific heat, the enthalpy is instead calculated with the NASA polynomial [27, p. 9] shown in Eq. (3.7), with coefficients taken from the CHEMKIN table.

\[
\frac{H}{RT} = a_1 + a_2 \cdot \frac{T}{2} + a_3 \cdot \frac{T^2}{3} + a_4 \cdot \frac{T^3}{4} + a_5 \cdot \frac{T^4}{5} + a_6 \cdot \frac{1}{T}
\]  
(3.7)
Thermal conductivity

It was seen in Table 3.2 that the thermal conductivity was calculated with the same thermophysical function as the enthalpy. This was not desired, thus a new class NSRDSfunc_k was created. The thermal conductivity was changed to be calculated with a polynomial of 3rd order, fitted to the tabled values in NIST table [28]. Fig. 3.3 shows the plotted thermal conductivities to their respective temperature from the NIST table.

![Figure 3.3: Polynomial of thermal conductivity](image)

A polynomial was fitted to the data using polyfit in Matlab. The equation takes the form:

\[ k = a_1 \cdot T^3 + a_2 \cdot T^2 + a_3 \cdot T + a_4 \]  

(3.8)

The coefficients were determined and are shown in Table 3.6.

| \(a_1\) | \(-0.0000001065\) |
| \(a_2\) | \(0.0002456\) |
| \(a_3\) | \(0.03224\) |
| \(a_4\) | \(5.894\) |

Table 3.6: Coefficients for thermal conductivity polynomial

Surface tension

Surface tension is not a factor for a gaseous sphere because it is a property of a liquid. The function remained unchanged because, as seen in Table 3.2, the surface tension is calculated with the same thermophysical function as the latent heat, thus this is already set to zero.

Gas properties

As the modifications to the liquid properties consist of treating it as a gas, no special care is needed for the gas properties shown in Table 3.1. All gas properties were changed for the fuel
myCH4 to be calculated with the same thermophysical function as the corresponding parcel property.

**Unchanged properties**

The second virial coefficient and vapour diffusivity coefficient were assumed to not have any noticeable impact to the simulations with regards to other assumptions and limitations taken in the thesis. Thus no changes were made to these properties or thermophysical functions.

### 3.3.2 Modifications to the evaporation characteristics

As found out in Section 2.2, the evaporation should occur instantaneously after a parcel reach a certain distance downstream, \( X_{\text{core}} \). A new evaporation model named mystandardEvaporation-Model was created. A core region was defined as a cylindrical shape with the length \( X_{\text{core}} \) and radius \( r_{\text{noz}} \) and is shown schematically in Fig. 3.4, where it is represented by the blue-striped region. Fig. 3.4 also shows a schematic representation of the spray angle \( \alpha \). Possibly, the core region could be defined as a conical shape to better represent the jet core, but the cylindrical core area was used in the gaseous LPT model because it was used in Hessel et al. [4].

The strategy for implementing was that the evaporation model is turned off for a parcel located in the core region and switched on if a parcel leaves. When a parcel leaves the core region it is instantaneously removed and its mass, momentum and energy is added to the gas phase in the computational cell in which the parcel evaporates. Details of the implementation of the evaporation characteristics is found in Appendix C and the full evaporation model is presented in Appendix D.

![Figure 3.4: Schematic representation of the core region](image.png)
A few simulations were performed to test the implementation of evaporation characteristics. The gaseous sphere penetration length for various values of $X_{\text{core}}$ are shown in Fig. 3.5. The gaseous sphere penetration is following the set value of $X_{\text{core}}$ for all simulations, which proves that the modified evaporation characteristics is working as intended. The sudden dip in penetration that occur from time to time can be explained by that a few parcels are injected at an angle that makes the parcel leave the core area before it reaches $X_{\text{core}}$.

![Figure 3.5: Gaseous sphere penetration for varying $X_{\text{core}}$.](image)

### 3.3.3 Compilation instructions

To be able to run dieselFoam with the modifications above, the new libraries and solver had to be compiled with custom settings. The compilation settings are divided into two files contained in the Make folder, called files and options. Instructions for compiling the new solver and libraries are briefly presented in Appendix E. Complete tutorials for compiling a new fuel, libraries and solver can be found in [29,30].
Chapter 4

Case setup and key parameters

4.1 Case setup

4.1.1 Geometry and mesh

As stated in Section 1.3, a 2D-axisymmetric geometry was used to represent a constant volume chamber. Technically OpenFOAM only operate in 3D. 2D is represented by a three-dimensional wedge shape with a thickness of one cell and a certain wedge angle, as shown in Fig. 4.1. The mesh was generated using the blockMesh utility, which is part of the standard OpenFOAM package.

![Figure 4.1: Wedge shaped geometry](image)

Figure 4.2 shows a schematic view of the geometry. The inlet is represented by a patch at the bottom left corner of the domain with a height representative of the nozzle radius. The rotational-axis is located at the bottom and the rest of the chamber is surrounded by walls. Below in Table 4.1, the dimensions for the geometry are listed. The mesh consisted of a structured grid with 40 hexahedra elements in the y-direction and and 100 in the x-direction (100x40), a total of 4000 computational cells. The mesh was scaled and refined close to the nozzle so that the smallest cells were in the order of 20% of the nozzle radius.
### 4.1.2 Boundary and initial conditions

Several cases were set up with varying boundary and initial conditions. A no-slip condition with wall functions was used for the velocity at the walls and a zero-gradient condition for temperature and pressure. The jet was said to create its own turbulence, meaning that $k$ and $\epsilon$ were initially set to a small value both in the domain and at the inlet. The injection duration was uniform for all cases at 4 ms and for all cases, 100% methane was injected into 100% nitrogen. Simulations were performed for varying inlet velocities and chamber conditions and the cases are listed in Table 4.2.

<table>
<thead>
<tr>
<th>Case</th>
<th>$U_{x,\text{noz}}$ (m/s)</th>
<th>$T_{ch}$ (K)</th>
<th>$p_{ch}$ (bar)</th>
<th>$T_f$ (K)</th>
<th>$m_f$ (g/s)</th>
<th>$m_f$ (mg)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>100</td>
<td>1000</td>
<td>10</td>
<td>373</td>
<td>1.625</td>
<td>6.500</td>
</tr>
<tr>
<td>Case 2</td>
<td>200</td>
<td>1000</td>
<td>10</td>
<td>373</td>
<td>3.250</td>
<td>13.000</td>
</tr>
<tr>
<td>Case 3</td>
<td>500</td>
<td>1000</td>
<td>10</td>
<td>373</td>
<td>8.125</td>
<td>32.500</td>
</tr>
<tr>
<td>Case 4</td>
<td>100</td>
<td>1000</td>
<td>40</td>
<td>373</td>
<td>6.500</td>
<td>26.000</td>
</tr>
</tbody>
</table>

Table 4.2: Boundary and initial conditions for the cases

### 4.1.3 Injector types and models

In dieselFoam there are a couple of standard injector types to choose from. It includes for example common rail injector, unit injector and defined injector. The injectors differ from each other in the way they are set up and mainly how they calculate the inlet velocity. The common rail injector type is the only chamber pressure dependent injector and the velocity is calculated
according to Eq. (4.1).

\[ U = \sqrt{\frac{2(P_{inj} - P_{amb})}{\rho}} \]  

The unit injector calculates the velocity from the mass flow as shown in Eq. (4.2), where \( C_D \) is a discharge coefficient and not the drag coefficient.

\[ U = \frac{\dot{m}}{\rho C_D A} \]  

The velocity for the defined injector is given as a velocity profile. To get a good comparison between the gaseous LPT method and the fully resolved jet a defined injector was chosen. Position, direction, nozzle diameter, total injected mass, fuel temperature, number of parcels, velocity profile and mass flow rate profile has to be specified for the defined injector. All velocity- and mass flow rate profiles were estimated with a square profile with constant values for the entire injection duration. The injector was placed at origin, i.e. the bottom left corner of the domain. Mass flow and total injected mass were estimated with Eqs. (4.3) and (4.4). The number of injected parcels varied for the cases but were in the order of 10 000. This was generally set merely to avoid crashes of the simulation.

\[ \dot{m}_f = \rho_f \cdot U_x \cdot A_n \]  

\[ m_f = \dot{m}_f \cdot t_{inj} \]  

An injector model also has to be selected. The injector model describes the initial droplet size and cone angle of the spray. There are several models in the standard OpenFOAM package and two of them are hollow cone injector and constant injector. For the hollow cone injector, the initial droplet size is calculated by a probability density function, which estimates droplet size in a statistical way. The spray angle is defined by specifying an inner and outer cone angle, which can either be constant or vary over time. For the constant injector model, an initial droplet size as well as spray angle is a constant specified value. The constant injector model was chosen and droplet size was set a constant size of 200 \( \mu m \). A sensitivity study of the droplet size will be presented in Chapter 5. The spray angle was set to 17\(^o\) and it was retrieved from measuring the gas jet expansion angle from the fully resolved simulation of Case 1. \( X_{core} \) was set at a constant value of \( X_{core} = 12.5 \, mm \), which was the theoretical value suggested in Section 2.2.

### 4.2 Key parameters

With the implementation of the gaseous LPT approach, some new parameters were introduced as part of the spray. As already introduced they were \( X_{core} \), initial droplet size, number of parcels and spray angle. To properly evaluate the LPT approach, the flow was investigated by studying the velocity and mass fraction at the center axis and over a few cross-sections at different locations downstream.

The newly introduced parameters and their effects on the gas jet were also investigated. A sensitivity analysis was performed by running Case 1 with varying \( X_{core} \), droplet size, spray angle and number of parcels. The effects of the grid was also briefly investigated by simulating Case 1 with a few different meshes with varying amount of cells.
4.3 Post-processing

Data was retrieved with the sample utility in standard OpenFOAM. A horizontal line at the center axis and cross-sections at positions \(x = 20\, mm\), \(x = 50\, mm\) and \(x = 80\, mm\) were defined. Velocity and methane mass fraction were sampled at these lines for all saved time steps. This data was then processed in Matlab. No GUI (Graphical User Interface) is available in OpenFOAM; hence, Paraview was used to graphically view the results and create contour plots.
Chapter 5

Results and discussion

Figs. 5.1 and 5.2 shows contour plots for methane mass fraction of Case 1, simulated with the gaseous LPT method and the fully resolved RANS method respectively. Plots are shown for the time steps 1, 2, 3 and 4 ms in the simulations and the white line represents a methane mass fraction of 3%. It appears that the gaseous LPT method generates a smaller radial spread compared with the resolved method and a longer vapour penetration for all time steps. Similar contour plots for Case 2, 3 and 4 are not presented because they all follow the same trend. They can however be found in Appendix F.

Figure 5.1: Contour plots of mass fraction for various time steps with LPT method for Case 1
Vapour penetration length was plotted for Case 1 with both the gaseous LPT and fully resolved method and are shown together with Eq. 2.28 in Fig. 5.3. It shows that both methods agree well with Eq. 2.28 which is based on real gas jet experiments. Eq. 2.28 is used to give an indication how representative the simulation is of a real jet. No further effort was spent on the mesh, setup and solution parameters since the comparison between gaseous LPT method and fully resolved RANS method, with the same mesh and other setup conditions, is the most interesting and relevant. The resolved case clearly under-predicts the penetration length compared to Eq. 2.28 and it is expected because RANS modelling is known to over-predict gas jet diffusion. The gaseous LPT method appears to be more in line with Eq. (2.28), however as the gaseous LPT method is also based on RANS turbulence modelling it is expected the same behaviour as the resolved case. This gives a first indication that the gaseous LPT method appears to under-predict the radial spreading and over-predict penetration length.
The vapour penetration was plotted for all cases with the gaseous LPT- and fully resolved method and are shown in Fig. 5.4. The figure further confirms the previous observation that the gaseous LPT method appears to over-predict penetration length as compared with the resolved method. It shows similar behaviour for all cases. The difference in penetration is in the order of 5 - 10 mm as the jet approaches the back wall of the chamber. However, the gaseous LPT method follows the trends of the fully resolved method for all cases, which suggests that it is possible model direct gas injection with the gaseous LPT approach and gain reasonable results.

Figure 5.3: Vapour penetration over time for Case 1
Properties at the center axis were investigated and Fig. 5.5 shows the centerline mass fraction for the previously chosen time steps. Fig. 5.5 only shows the mass fraction of the gas phase and that is the reason why for $X < X_{\text{core}}$ the mass fraction is zero. This part is however filled with gaseous parcels and if this is taken into consideration it can be thought of as the jet core. At $X = X_{\text{core}}$, the mass fraction rapidly increases because all gaseous parcels evaporate at this location, reaching almost unity. All graphs agree that the mass fraction follows the trends of the resolved method but the penetration length is in general longer with the gaseous LPT method. The decay of mass fraction after the core is significantly slower for Case 1 and 4 whereas for Case 2 and 3, where the inlet velocities were increased, the gaseous LPT method appears to reproduce the gas jet behaviour in this region better.
Similarly, the centerline velocities were plotted and shown in Fig. 5.6. The axial velocity is scaled with the inlet velocity. An observation one makes from Fig. 5.6 is the phenomenon which appears between $x/r = 10$ and 20. The velocity rapidly decreases to reach a minimum value and suddenly increases again to reach a maximum value before it settles. This behaviour appears for all cases and the amplitude of this phenomenon seem to vary not only between the cases but also over time. After the velocities reaches its peak after $X_{core}$, the velocity starts to decrease due to the jet expansion. The decay of velocity from the LPT approach is slower than that for the resolved method and eventually it reaches roughly the same value as with the resolved method.
Figure 5.6: Centerline velocity for various time steps

This oscillatory velocity profile at $X_{\text{core}}$ was investigated more thoroughly as it is a behaviour introduced with the gaseous LPT method. It was believed that the centerline velocity fluctuations are a result of the instantaneous evaporation that is occurring at $X_{\text{core}}$. When a gaseous sphere evaporates all of its mass is added to the gas phase in the cell in which it evaporates. A sudden change of density occurs locally which could affect the velocity in the surroundings. The centerline gas pressure was investigated for a few arbitrary time steps close to each other and is shown in Fig. 5.7. The sudden change of pressure is clearly observed, a high pressure peak is present at $X_{\text{core}}$ and it appears to fluctuate and dip below the chamber pressure. It is intuitive that the velocity in close surroundings is affected.
A plausible explanation to this behaviour is be that, when a gaseous parcel, travelling with the velocity $U$ evaporates, the mass of the parcel leaves with a velocity $u'_s$ in all directions. This contributes to a deceleration backwards and an acceleration forward of the gas phase from the location of evaporation. This is illustrated in Fig. 5.8.
The radial values of mass fraction and velocity are presented at three different positions downstream for 1 ms after the start of injection. This time was chosen because at this time the jets of all cases are at different stages, as seen in Fig. 5.4. Figs. 5.9 and 5.10 give a clearer view of the lower spreading, predicted with the gaseous LPT method. Comparing the results from the different measurement locations, it can be seen that further downstream the difference in spreading is larger. The mass fraction and velocities are scaled with at the corresponding centerline values for respective location.

Figure 5.9: Radial mass fraction profiles for various locations downstream
Figure 5.10: Radial velocity profiles for various locations downstream
5.1 Sensitivity analysis

The sensitivity analysis was performed to get an understanding of how the parameters introduced with the gaseous LPT method affects the penetration and spreading of the jet. Grid size, $X_{core}$, droplet size and parcel number all have to be chosen in consistence with each other, which is inherent to any LPT method. Thus an direct evaluation of individual parameters is difficult. With better knowledge and more care regarding these parameters the results could have been improved. The sensitivity analysis was performed with the base setup for Case 1.
Influence of $X_{\text{core}}$

Fig. 5.11 shows the vapour penetration for simulations run with different $X_{\text{core}}$ values. It can be seen that in general, they follow the same trend and a higher value of $X_{\text{core}}$ generates a longer vapour penetration. This is in agreement with observations made by Hessel et al. [4]. Fig. 5.12 shows the radial velocity- and mass fraction profiles at $x/r = 80$, 4 ms after the start of injection. Increasing $X_{\text{core}}$ seem to not only increase the vapour penetration but also the radial spreading. It appears that by decreasing $X_{\text{core}}$, the LPT simulated vapour penetration will be in close agreement with the resolved results; however, the spread in the radial direction will be in greater difference with those from the resolved method, cf. Figs. 5.3 and 5.9.

Figure 5.11: Vapour penetration for varying $X_{\text{core}}$

Figure 5.12: Radial profiles at $x/r = 80$ for varying $X_{\text{core}}$
Influence of grid size

Fig. 5.13 shows the vapour penetration for a few simulations run with varying grid size. Simulations were carried out with grids of 150x40, 100x40, 80x40 and 80x25 cells. It can be seen that the penetration varies for the different grids up to an order of 10 mm between the cases after four seconds of injection time. The greatest difference is for the coarse grid with fewer cells in radial direction, which resulted in a slower penetration rate towards the end of injection. Fig. 5.14 shows the radial velocity- and mass fraction profiles at $x/r = 80$, 4 ms after start of injection. The radial spreading is also grid sensitive, a coarser mesh appears to predict a lower spreading.

![Vapour penetration for varying grid size](image1)

Figure 5.13: Vapour penetration for varying grid size

![Radial profiles at x/r = 80 for varying grid size](image2)

Figure 5.14: Radial profiles at $x/r = 80$ for varying grid size
Influence of parcel number

Fig. 5.15 shows the penetration length for three simulations performed with a different number of injected parcels. As the same amount of mass is injected per time step, a lower number of parcels result in a higher parcel mass. It can be seen that varying the parcel number also affects the vapour penetration. It appears that a lower amount of parcels yields a slightly lower vapour penetration, but it only differ marginally between 7000 and 5000 parcels. Fig. 5.16 shows the radial velocity- and mass fraction profiles at $x/r = 80$, 4 ms after the start of injection. It shows that the parcel number does not have any significant influence on the radial spreading.

Figure 5.15: Vapour penetration for varying parcel number

Figure 5.16: Radial profiles at $x/r = 80$ for varying parcel number
Influence of gaseous sphere size

Fig. 5.17 shows the penetration length of simulations run with varying gaseous sphere size. It appears that the gaseous sphere size does not have any noticeable affect on the penetration length. Fig. 5.18 shows the radial velocity- and mass fraction profiles at $x/r = 80$, 4 ms after start of injection. The influence of gaseous sphere size to the radial spreading appears to be insignificant.

![Figure 5.17: Vapour penetration for varying gaseous sphere size](image)

![Figure 5.18: Radial profiles at $x/r = 80$ for varying gaseous sphere size](image)
Influence of spray angle

Fig. 5.19 shows the vapour penetration and Fig. 5.20 shows the radial velocity- and mass fraction profiles at $x/r = 80$. Simulations were performed with two different spray angles, 17 and 25 degrees. Both figures show that the spray angle has no significant influence on either the vapour penetration or the radial spreading.

Figure 5.19: Vapour penetration for varying spray angle

Figure 5.20: Radial profiles at $x/r = 80$ for varying spray angle
6.1 Conclusions

It was shown in this thesis that the existing LPT solver dieselFoam in OpenFOAM 2.0.x, with a few modifications, can be used to model subsonic gas injection. The gaseous LPT method provides similar trends as observed in the resolved simulations. However, in general the method under-predicts the radial spreading of the jet and results in greater vapour penetration. This phenomenon was observed for all cases and agrees with observations made by Hessel et al. [5]. It is believed to be an effect of the instantaneous evaporation occurring at a specific location which is causing local pressure fluctuations at $X_{\text{core}}$. When the mass of all parcels is added to the gas phase it is influencing the surrounding velocity and pressure fields.

The results from the sensitivity analysis showed that the simulations were sensitive to the grid size and the core length, but less sensitive to the spray angle, the number of injected gas parcels and the gaseous sphere sizes. It is possible that further work on the choice of injection parameters could lead to improved results. Despite these observations, the LPT approach in OpenFOAM can be used to model gas injection and gain reasonable results. It was not in the scope of this thesis to optimize these parameters and this is left for future work.

6.2 Suggestions for future work

For further work of developing and evaluating the gaseous LPT model, the following are suggested:

- Further investigate the dependency of the injection parameters. Full understanding of the parameters is essential to obtain a robust simulation method.

- Improve the transition phase from parcels to the gas phase. Possibly, a statistical approach with a variance could be used to define $X_{\text{core}}$ to smoothen the transition.

- Perform simulations on 3D meshes to evaluate the full potential regarding cell requirement and computational time.

- Validate the model against experimental data.
References


Appendix A

Source code for the implemented fuel

Constructor of myCH4.C

```cpp
#include "myCH4.H"
#include "addToRunTimeSelectionTable.H"

namespace Foam {
    defineTypeNameAndDebug(myCH4, 0);
    addToRunTimeSelectionTable(liquidProperties, myCH4,);
    addToRunTimeSelectionTable(liquidProperties, myCH4, Istream);
    addToRunTimeSelectionTable(liquidProperties, myCH4, dictionary);
}

namespace Foam {
    liquidProperties()
    {
        16.043, // W
        GREAT, // Tc
        GREAT, // Pc
        GREAT, // Vc
        GREAT, // Zc
        SMALL, // Tt
        SMALL, // Pt
        GREAT, // Tb
        0.0, // dipm
        0.0403, // omega
        9.0819e+3 // delta
    },
    rho_(518.3),
    pw_(),
    hl_(540.20, 499121.791545248, 0.38795, 0.0, 0.0, 0.0),
    Cp_()
    {
        0.01683479E+02, // a1_high
        0.01023724E+00, // a2_high
        -0.03875129E-04, // a3_high
        0.06785585E-08, // a4_high
        -0.04503423E-12, // a5_high
        -0.01008079E+06, // b1_high
        0.09623395E+02, // b2_high
    }
}
```
0.07787415E+01, //a1_low
0.01747668E+00, //a2_low
−0.02783409E−03, //a3_low
0.03049708E−06, //a4_low
−0.01223931E−09, //a5_low
−0.09825229E+05, //b1_low
0.01372219E+03, //b2_low
518.3, //Specific gas constant methane
1000 //Tcommon
)

h_
(}
0.01683479E+02, //a1_high
0.01023724E+00, //a2_high
−0.03875129E−04, //a3_high
0.06785585E−08, //a4_high
−0.04503423E−12, //a5_high
−0.01008079E+06, //b1_high
0.09623395E+02, //b2_high
0.07787415E+01, //a1_low
0.01747668E+00, //a2_low
−0.02783409E−03, //a3_low
0.03049708E−06, //a4_low
−0.01223931E−09, //a5_low
−0.09825229E+05, //b1_low
0.01372219E+03, //b2_low
518.3, //Specific gas constant methane
1000 //Tcommon
)

Cpg_
(}
0.01683479E+02, //a1_high
0.01023724E+00, //a2_high
−0.03875129E−04, //a3_high
0.06785585E−08, //a4_high
−0.04503423E−12, //a5_high
−0.01008079E+06, //b1_high
0.09623395E+02, //b2_high
0.07787415E+01, //a1_low
0.01747668E+00, //a2_low
−0.02783409E−03, //a3_low
0.03049708E−06, //a4_low
−0.01223931E−09, //a5_low
−0.09825229E+05, //b1_low
0.01372219E+03, //b2_low
518.3, //Specific gas constant methane
1000 //Tcommon
)

B_
(}
0.00274040956448844,
−2.90407568560137,
−410900.562851782,
−8.78208454752305e+17,
1.28238393676899e+20
)

mu_
(}
1.578968e02, //S
1.0124299e−06 //C1
)

mug_
(}
1.578968e02, //S
1.0124299e-06 //Cl

K_{(-0.0000010653968, 0.000245678004, 0.032246337057742, 5.894846729933392)},
K_{g_{(-0.0000010653968, 0.000245678004, 0.032246337057742, 5.894846729933392)}},
sigma_{(540.20, 0.054143, 1.2512, 0.0, 0.0, 0.0)},
D_{(147.18, 20.1, 16.043, 28.0)}

// ****************************************** //
Appendix B

Source code for the modified thermophysical functions

NSRDSfunc0.H

```c
/*---------------------------------------------*
 | Class NSRDSfunc0 Declaration               |
 *---------------------------------------------*/

class NSRDSfunc0
{
    public thermophysicalFunction
{
    // Private data

    // NSRDS function 100 coefficients
    scalar a_, b_, c_, d_, e_, f_, a1_low_, a2_low_, a3_low_, a4_low_, a5_low_,
    a6_low_, a7_low_, a1_high_, a2_high_, a3_high_,
    a4_high_, a5_high_, a6_high_, a7_high_, R_, Tcommon_

    public:

    // Runtime type information
    TypeName( 'NSRDSfunc0' );

    // Constructors

    // Construct from components
    NSRDSfunc0
    (const scalar a,
     const scalar b,
     const scalar c,
     const scalar d,
     const scalar e,
     const scalar f
    );

    // Construct from components
    NSRDSfunc0
    (const scalar a1_high,
     const scalar a2_high,
     const scalar a3_high,
```
const scalar a4_high,
const scalar a5_high,
const scalar a6_high,
const scalar a7_high,
const scalar a1_low,
const scalar a2_low,
const scalar a3_low,
const scalar a4_low,
const scalar a5_low,
const scalar a6_low,
const scalar a7_low,
const scalar R,
const scalar Tcommon

NSRDSfunc0(Istream& is);
NSRDSfunc0(const dictionary& dict);

// Member Functions
// Evaluate the function and return the result
scalar f(scalar, scalar T) const
{
    if (T < Tcommon_){
        return R_ * (a1_low_ * T + a2_low_ * pow(T, 2) / 2 + a3_low_ * pow(T, 3) / 3 +
                     a4_low_ * pow(T, 4) / 4 + a5_low_ * pow(T, 5) / 5 + a6_low_);
    } else{
        return R_ * (a1_high_ * T + a2_high_ * pow(T, 2) / 2 + a3_high_ * pow(T, 3) / 3 +
                     a4_high_ * pow(T, 4) / 4 + a5_high_ * pow(T, 5) / 5 + a6_high_);
    }
}
/ Constructors

// Construct from components
NSRDSfunc1
{
    const scalar S,
    const scalar C1
};

// Construct from components
NSRDSfunc1
{
    const scalar a,
    const scalar b,
    const scalar c,
    const scalar d,
    const scalar e
};

// Construct from Istream
NSRDSfunc1(Istream& is);

// Construct from dictionary
NSRDSfunc1(const dictionary& dict);

// Member Functions

// Evaluate the function and return the result
scalar f(scalar p, scalar T) const
{
    return C1_*pow(T,1.5)/(T+S_);
}

NSRDSfunc5.H

/*---------------------------------------------*/
| Class NSRDSfunc5 Declaration |
/*---------------------------------------------*/

class NSRDSfunc5
:
public thermophysicalFunction
{
    // Private data

    // NSRDS function 105 coefficients
    scalar a_, b_, c_, d_, R_;

public:

    // Runtime type information
    TypeName('NSRDSfunc5');

    // Constructors

    // Construct from components
    NSRDSfunc5
(     const scalar a,
    const scalar b,
    const scalar c,
    const scalar d
    );

    //− Construct from components
    NSRDSfunc5
    (     const scalar R
    );

    //− Construct from Istream
    NSRDSfunc5(Istream& is);

    //− Construct from dictionary
    NSRDSfunc5(const dictionary& dict);

    // Member Functions

    //− Evaluate the function and return the result
    scalar f(scalar p, scalar T) const
    {
      return p/(R_*T);
    }

NSRDSfunc6.H

/*@-------------------------------*/

Class NSRDSfunc6 Declaration
/*@-------------------------------*/

class NSRDSfunc6
 :
  public thermophysicalFunction
{
  // Private data

    // NSRDS function 106 coefficients
    scalar Tc_, a_, b_, c_, d_, e_;

  public:

    //− Runtime type information
    TypeName("NSRDSfunc6");

    //− Constructors

    //− Construct from components
    NSRDSfunc6
    (     const scalar Tc,
        const scalar a,
        const scalar b,
        const scalar c,
        const scalar d,
        const scalar e
    );
// Construct from Istream
NSRDSfunc6(Istream& is);

// Construct from dictionary
NSRDSfunc6(const dictionary& dict);

// Member Functions

// Evaluate the function and return the result
scalar f(scalar, scalar T) const
{
    return SMALL;
}

NSRDSfunc14

class NSRDSfunc14 : public thermophysicalFunction
{
    // Private data

    // NSRDS function 114 coefficients
    scalar Tc_, a_, b_, c_, d_, a1_high_, a2_high_, a3_high_, a4_high_,
    a5_high_, a6_high_, a7_high_,
    a1_low_, a2_low_, a3_low_, a4_low_, a5_low_, a6_low_, a7_low_, R_,
    Tcommon_;

public:

    // Runtime type information
    TypeName("NSRDSfunc14");

    // Constructors

    // Construct from components
    NSRDSfunc14(
        const scalar Tc,
        const scalar a,
        const scalar b,
        const scalar c,
        const scalar d
    );

    // Construct from components
    NSRDSfunc14(
        const scalar a1_high,
        const scalar a2_high,
        const scalar a3_high,
        const scalar a4_high,
        const scalar a5_high,
        const scalar a6_high,
    );
const scalar a7_high,
const scalar a1_low,
const scalar a2_low,
const scalar a5_low,
const scalar a4_low,
const scalar a5_low,
const scalar a6_low,
const scalar a7_low,
const scalar R,
const scalar Tcommon
);

// Construct from Istream
NSRDSfunc14(Istream& is);

// Construct from dictionary
NSRDSfunc14(const dictionary& dict);

// Member Functions
// Evaluate the function and return the result
scalar f(scalar, scalar T) const
{
  if (T < Tcommon_){
    return R_ * (((a5_low_*T + a4_low_)*T + a3_low_)*T + a2_low_)*T + a1_low_);
  } else{
    return R_ * (((a5_high_*T + a4_high_)*T + a3_high_)*T + a2_high_)*T + a1_high_);
  }
}

NSRDSfunc_k

/*
 *-------------------------------
 * Class NSRDSfunc_k Declaration
 *-------------------------------*
*/
class NSRDSfunc_k :
  public thermophysicalFunction
{
  // Private data
  // NSRDS function 100 coefficients
  scalar a_, b_, c_, d_, e_, f_, a1_, a2_, a3_, a4_;

  public:

  // Runtime type information
  TypeName("NSRDSfunc_k");

  // Constructors
  // Construct from polynomial coefficients
  NSRDSfunc_k
  (}
const scalar a1,
const scalar a2,
const scalar a3,
const scalar a4
);

// Construct from components
NSRDSfunc_k(const scalar a,
            const scalar b,
            const scalar c,
            const scalar d,
            const scalar e,
            const scalar f);

// Construct from Istream
NSRDSfunc_k(Istream& is);

// Construct from dictionary
NSRDSfunc_k(const dictionary& dict);

// Member Functions

// Evaluate the function and return the result
scalar f(scalar, scalar T) const
{
    return a1_*pow(T,3) + a2_*pow(T,2) + a3_*T + a4_;
}

NSRDSfunc_pv

/*-------------------------------------*/
Class NSRDSfunc_pv Declaration
/*-------------------------------------*/

class NSRDSfunc_pv
:
    public thermophysicalFunction
{
    // Private data

    // NSRDS function 100 coefficients
    scalar a_, b_, c_, d_, e_, f_ ;

    public:

    // Runtime type information
    TypeName("NSRDSfunc_pv") ;

    // Constructors

    // Construct from nothing
    NSRDSfunc_pv() ;

    // Construct from components
    NSRDSfunc_pv(const scalar a,
const scalar b,
const scalar c,
const scalar d,
const scalar e
);
// Construct from Istream
NSRDSfunc_pv(Istream& is);

// Construct from dictionary
NSRDSfunc_pv(const dictionary& dict);

// Member Functions

// Evaluate the function and return the result
scalar f(scalar p, scalar T) const
{
    return SMALL;
}
Appendix C

Modifications to the parcel class

Some changes had to be done to the parcel class. In parcel.C the code checks if evaporation is active at several locations. This is done with the line below.

```cpp
if (sDB.evaporation().evaporation())
```

It is necessary for the evaporation model to access the parcel position, thus every occasion of this line was changed to,

```cpp
if (sDB.evaporation().evaporation(position().x(), position().y()))
```

where position().x() and position().y() input the respectively x- and y-position of the current parcel to the evaporation model.

Additionally a piece of code in parcel.C checks if the parcel has reached critical condition, which originally is if the droplet temperature has reached the critical temperature. If the condition is met the parcel is immediately removed and added to the gas phase. The code looks like:

```cpp
// immediately evaporate mass that has reached critical
// condition
if (mag(Tnew - fuels.Tc(X()))) < SMALL)
{
    mi[i] = 0.0;
}
```

The criteria of critical condition was changed to match the desired behaviour, to evaporate if the parcel leaves the core region. The code was changed into:

```cpp
// immediately evaporate mass that has reached critical
// condition
if (position().x() > sDB.evaporation().Xcore() || position().y() > sDB.evaporation().Ycore())
{
    mi[i] = 0.0;
}
```
Appendix D

Evaporation model

mystandardEvaporationModel.C

```c
#include "error.H"
#include "mystandardEvaporationModel.H"
#include "addToRunTimeSelectionTable.H"

namespace Foam {
    defineTypeNameAndDebug(mystandardEvaporationModel, 0);
    addToRunTimeSelectionTable(
        evaporationModel, mystandardEvaporationModel, dictionary)
}

namespace Foam {
    mystandardEvaporationModel::mystandardEvaporationModel(
        const dictionary& dict)
    :
        evaporationModel(dict),
        evapDict_(dict.subDict(typeName + "Coeffs")),
        preReScFactor_(readScalar(evapDict_.lookup("preReScFactor"))),
        ReExponent_(readScalar(evapDict_.lookup("ReExponent"))),
        ScExponent_(readScalar(evapDict_.lookup("ScExponent"))),
        evaporationScheme_(evapDict_.lookup("evaporationScheme")),
        Xcore_(readScalar(evapDict_.lookup("Xcore"))),
        Ycore_(readScalar(evapDict_.lookup("Ycore"))),
        nEvapIter_(0)
    {
        if (evaporationScheme_ == "implicit")
        {
            nEvapIter_ = 2;
        }
        else if (evaporationScheme_ == "explicit")
        {
            // Additional code
        }
    }
} // namespace Foam
```
nEvapIter_ = 1;
}
else {
    FatalError
    "evaporationScheme\_type\_" << evaporationScheme_
    ",unknown.\_Use\_Implicit\_or\_Explicit\_" << nl
    "abort(FatalError);"
}
const scalar rhoFuelVapor,
const scalar massDiffusionCoefficient,
const scalar ReynoldsNumber,
const scalar SchmidtNumber,
const scalar Xs,
const scalar Xf,
const scalar m0,
const scalar dm,
const scalar dt
) const
{
    return SMALL;
}

Foam:: scalar Foam::mystandardEvaporationModel::boilingTime
(
    const scalar liquidDensity,
    const scalar cpFuel,
    const scalar heatOfVapour,
    const scalar kappa,
    const scalar Nusselt,
    const scalar deltaTemp,
    const scalar diameter,
    const scalar,
    const scalar,
    const scalar,
    const scalar,
    const scalar,
    const scalar,
    const scalar,
    const scalar,
    const scalar)
) const
{
    return GREAT;
}

// ************************************************************************//
Appendix E

Compilation instructions

Compile thermophysicalFunctions

The following lines were added to the list of functions in Make/files to include the created thermophysical functions.

\[
\$(NSRDSfunctions)/NSRDSfunc_pv/NSRDSfunc_pv.C \\
\$(NSRDSfunctions)/NSRDSfunc_k/NSRDSfunc_k.C
\]

Additionally the line at the bottom of Make/files was changed from

\[
LIB = \$(FOAM_LIBBIN)/libThermophysicalFunctions
\]
to

\[
LIB = \$(FOAM_USER_LIBBIN)/libmyThermophysicalFunctions
\]

Compile liquidProperties

The new fuel, myCH4 was added to Make/files as below

\[
liquidProperties/liquidProperties.C \\
myC7H16/myC7H16.C
\]

and the last line was changed from

\[
LIB=\$(FOAM_LIBBIN)/libLiquidProperties
\]
to

\[
LIB=\$(FOAM_USER_LIBBIN)/libmyLiquidProperties
\]

The file Make/options was changed from

\[
EXE_INC= \$
    -I$(LIB_SRC)/thermophysicalModels/thermophysicalFunctions/lnInclude
\\
LIB_LIBS= \$
    -lthermophysicalFunctions
\]
to

\[
EXE_INC= \$
    -I$(WM_PROJECT_USER_DIR)/src/thermophysicalModels/thermophysicalFunctions/lnInclude
\\
LIB_LIBS= \$
    $(FOAM_USER_LIBBIN)/libmyThermophysicalFunctions.so
\]
Compile dieselSpray library

The new evaporation model was added to Make/files by simply adding the line.

\$\{(evaporationModels)/mystandardEvaporationModel/mystandardEvaporationModel\}.C

The last line in Make/files was changed from

\[LIB = \$(FOAM_LIBBIN)/libdieselSpray\]

to

\[LIB = \$(FOAM_USER_LIBBIN)/libmydieselSpray\]

In Make/options the custom liquidProperties and thermophysicalFunctions libraries were added by changing the following lines from

\[-I\$(LIB_SRC)/thermophysicalModels/properties/liquidProperties/lnInclude \ \
-\$(LIB_SRC)/thermophysicalModels/thermophysicalFunctions/lnInclude \ \\
\]

to

\[-I\$(WM_PROJECT_USER_DIR)/src/thermophysicalModels/properties/liquidProperties/lnInclude \ \
-\$(WM_PROJECT_USER_DIR)/src/thermophysicalModels/thermophysicalFunctions/lnInclude \ \\
\]

And following was changed from

\[-lliquidProperties \ 
-thermophysicalFunctions \ \\
\]

to

\[\$(FOAM_USER_LIBBIN)/libmyLiquidProperties.so \ 
\$(FOAM_USER_LIBBIN)/libmyThermophysicalProperties.so \ \\
\]

Compile dieselFoam solver

Finally a new solver had to be compiled. The standard dieselFoam solver was copied and named mydieselFoam at local user directory. The code for the solver remained unchanged but the compile instructions had to be modified to read the correct modified libraries. This was done by changing the following line in Make/files from

\[EXE = \$(FOAM_APPBIN)/dieselFoam\]

to

\[EXE = \$(FOAM_USER_APPBIN)/mydieselFoam\]

And in Make/options the following was changed to make the solver read from the modified libraries from

\[-I\$(LIB_SRC)/lagrangian/dieselSpray/lnInclude \ \
-\$(LIB_SRC)/thermophysicalModels/properties/liquidProperties/lnInclude \ \
-\$(LIB_SRC)/thermophysicalModels/thermophysicalFunctions/lnInclude \ \\
\]

to

\[-I\$(WM_PROJECT_USER_DIR)/src/lagrangian/mydieselSpray/lnInclude \ \
-\$(WM_PROJECT_USER_DIR)/src/thermophysicalModels/properties/liquidProperties/lnInclude \ \
-\$(WM_PROJECT_USER_DIR)/src/thermophysicalModels/thermophysicalFunctions/lnInclude \ \\
\]

And further down

\[-ldieselSpray \ 
-lliquidProperties \ 
-llagrangian/dieselSpray/lnInclude \ \\
\]

was changed to

\[\$(FOAM_USER_LIBBIN)/libmyLiquidProperties.so \ 
\$(FOAM_USER_LIBBIN)/libmydieselSpray.so \ 
\$(FOAM_USER_LIBBIN)/libmyThermophysicalFunctions.so \]
Appendix F

Contour plots for Case 2, 3 and 4
Case 2

Figure F.1: Contour plots of mass fraction for various time steps with gaseous LPT method for Case 2

Figure F.2: Contour plots of mass fraction for various time steps with resolved method for Case 2
Case 3

Figure F.3: Contour plots of mass fraction for various time steps with gaseous LPT method for Case 3

Figure F.4: Contour plots of mass fraction for various time steps with resolved method for Case 3
Case 4

Figure F.5: Contour plots of mass fraction for various time steps with gaseous LPT method for Case 4

Figure F.6: Contour plots of mass fraction for various time steps with resolved method for Case 4