Cite as: Sula, C.: Implementation of a secondary droplet breakup model in OpenFOAM
In Proceedings of CFD with OpenSource Software, 2019, Edited by Nilsson. H.,
http://dx.doi.org/10.17196/OS_CFD#YEAR_2019

CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

Implementation of a secondary droplet breakup model in OpenFOAM

Developed for OpenFOAM-v1906

Author:
Sula Constantin
Université catholique de Louvain
constantin.sula@uclouvain.be

Peer reviewed by:
Mohammad Hossein Arabnejad
Yanyan Zhai

Disclaimer: This is a student project work, done as part of a course where OpenFOAM and some other OpenSource software are introduced to the students. Any reader should be aware that it might not be free of errors. Still, it might be useful for someone who would like learn some details similar to the ones presented in the report and in the accompanying files. The material has gone through a review process. The role of the reviewer is to go through the tutorial and make sure that it works, that it is possible to follow, and to some extent correct the writing. The reviewer has no responsibility for the contents.

April 22, 2020

Learning outcomes

The reader will learn:

How to use it:

- How to use the sprayFoam solver
- How to use lagrangian libraries

The theory of it:

- A general theory of spray flows with particles
- The theory of the Taylor Analogy Breakup models is discussed in detail

How it is implemented:

- A new secondary breakup model is implemented, by modifying an existing breakup model

How to modify it:

- A step-by-step tutorial on how to modify the Lagrangian libraries

Prerequisites

The reader is expected to know the following in order to get maximum benefit out of this tutorial:

- Fundamentals of Computation Fluid Dynamics and Lagrangian Particle Tracking method.
- Be able to run standard tutorials in OpenFOAM.
- Basic Linux knowledge.

Contents

Chapter 1

Introduction

Liquid-gas two-phase flows are frequently encountered in our daily lives, e.g. falling raindrops, and in industrial applications such as fuel injection in Diesel engines or gas turbines. In industry, the interactions occurring between liquids and gases are crucial for efficient operation. Therefore, the study of sprays, both numerically and experimentally has attracted a lot of attention. The multitude of processes occurring during spray atomization, such as breakup of the liquid jet, evaporation and mixing with the surrounding air, is quite complex. They are characterized by a multitude of spatial and temporal scales, which makes the entire process a challenging problem to study.

In this study, we will focus on the two-phase flows that typically occur during the injection of the liquid into a quiescent domain. Specifically, we focus on the flows that arise inside diesel engines, before combustion, as the fuel is injected under high pressure into the combustion domain. As the liquid fuel enters the domain, close to the nozzle, an intact liquid core can be distinguished close to the nozzle. Due to the interaction of the liquid with the surrounding air, the surface of the liquid core is destabilized. As a result, the liquid core breaks down, and droplets of different size are formed. This process is called the primary breakup of the jet. These droplets then interact further with the surrounding air and disintegrate into smaller droplets. The process of disintegration of the big droplets into small ones is called secondary breakup.

A very popular method to describe the flows numerically with particles is Lagrangian Particle Tracking (LPT), where the particles are tracked individually in a Lagrangian framework, while the gaseous phase is described in a Eulerian framework by the Navier-Stoke equations. As the droplets tend to have a spherical shape, tracking them individually reduces the computational cost. On the other hand, in the region of the liquid core, the liquid cannot be treated in a LPT framework. However, as this region is small compared to the length of the spray it is neglected in most numerical studies. Consequently, at the injection position droplets of different sizes are introduced into the computational domain. In order to decrease further the computational cost, particles with the same characteristics, (diameter, thermophysical characteristics and velocity) are grouped into *parcels*, and it is these parcels instead, that are introduced into the computational domain. Therefore, models are required to account for physical processes that occur at the droplet surface, like heat exchange, evaporation, collision, secondary breakup etc.

In this tutorial, we will focus on how the secondary breakup models are implemented in OpenFOAM, but the same philosophy applies and for the other phenomena that require models (e.g. evaporation, collision).

The manuscript follows the following structure: the second chapter gives a brief description of the physical problem and a small tutorial on how one can solve this problem numerically using a solver provided in OpenFOAM. The third chapter elaborates on the Lagrangian libraries. Specifically, how the parcels are created during the simulations, which libraries are related to the cloud object and how one can change the secondary breakup model reading the code. The fourth chapter first elaborates on the theory of the Taylor Analogy Breakup model and of the modification proposed herein. In the last chapter we present a step-by-step procedure on how to implement the new proposed sub-model as a new library. Finally, a case is set up using the newly implemented breakup model.

Chapter 2

Basic tutorial

2.1 aachenBomb

This section elaborates how to run the `aachenBomb` tutorial case using the `sprayFoam` solver. Also, we explain the solver output messages.

The sprayFoam solver is one of the solvers provided by OpenFOAM in order to simulate spray flows. Additionally, a basic tutorial for this solver, is provided under the directory:

```
$FOAM_TUTORIALS/lagrangian/sprayFoam/aachenBomb/
```

In order to run the tutorial, one needs first to copy it to the user directory by typing in the terminal:

```
cp -r $FOAM_TUTORIALS/lagrangian/sprayFoam/aachenBomb/ $FOAM_RUN/
cd $FOAM_RUN/aachenBomb
```

The tutorial contains the following folders:

```
0  chemkin  constant  system
```

The folder `0` contains the initial conditions for the gaseous fields, such as velocity, pressure, temperature, species mass fraction etc. The folder `chemkin` contains thermophysical properties of the species, reaction mechanism and transport properties for each species. The folder `constant` contains properties on how the chemistry (`chemistryProperties`) and the combustion (`combustionProperties`) during the simulations are numerically solved, turbulence properties (`turbulenceProperties`), thermophysical properties of the liquid components (`thermophysicalProperties`), radiation properties (`radiationProperties`), spray cloud properties (`sprayCloudProperties`) and once the mesh is generated a folder `polyMesh` is created under this directory. This latter directory contains all the information regarding the mesh. The system folder, contains the blockMesh dictionary (`blockMeshDict`), solver control dictionary (`controlDict`), discretization schemes (`fvSchemes`) and numerical methods dictionary (`fvSolution`) on how the systems are solved.

In order to run the case, one needs to generate the mesh using the command:

```
blockMesh
```

then run the solver in the background by typing:

```
sprayFoam >& log.sprayFoam&
```

If we open the `log.sprayFoam` file, you can find the following lines regarding the creation of the clouds, which represent a collection of parcels:

```
Constructing reacting cloud
Constructing particle forces
    Selecting particle force sphereDrag
```

```
Constructing cloud functions
    none
Constructing particle injection models
Creating injector: model1
Selecting injection model coneNozzleInjection
    Constructing 3-D injection
Selecting distribution model RosinRammler
Selecting dispersion model none
Selecting patch interaction model standardWallInteraction
Selecting stochastic collision model none
Selecting surface film model none
Selecting U integration scheme Euler
Selecting heat transfer model RanzMarshall
Selecting T integration scheme analytical
Selecting composition model singlePhaseMixture
Selecting phase change model liquidEvaporationBoil
Participating liquid species:
    C7H16
Selecting atomizationModel none
Selecting breakupModel ReitzDiwakar
Average parcel mass: 2.4e-10
Selecting radiationModel none
```

we also find thermophysical properties and the models regarding different physical phenomena of the parcels. All models are read from the dictionary when the cloud is constructed, and are read from the following file/dictionary:

```
$FOAM_RUN/aachenBomb/constant/sprayCloudProperties
```

At the start of each loop the the following message is displayed:

```
Solving 3-D cloud sprayCloud

Cloud: sprayCloud injector: model1
    Added 1 new parcels

Cloud: sprayCloud
    Current number of parcels       = 1
    Current mass in system          = 1.18387e-11
    Linear momentum                 = (-1.5539e-12 -9.73672e-11 -3.28379e-12)
   |Linear momentum|                = 9.7435e-11
    Linear kinetic energy           = 4.00955e-10
    Average particle per parcel     = 9.63933
    Injector model1:
      - parcels added               = 1
      - mass introduced             = 1.18421e-11
    Parcel fate: system (number, mass)
      - escape                      = 0, 0
    Parcel fate: patch (number, mass) walls
      - escape                      = 0, 0
      - stick                       = 0, 0
    Temperature min/max             = 334.164, 334.164
    Mass transfer phase change      = 3.36363e-15
    D10, D32, Dmax (mu)             = 15.3382, 15.3382, 15.3382
    Liquid penetration 95% mass (m) = 4.09432e-05
```

that denotes that the cloud is solved, specifically new parcels are added to the cloud, the position of the parcels are updated, also parcel properties are updated and the source terms for the exchange of mass, momentum and energy with the gaseous phase are calculated; and in the end some cloud properties are displayed. Afterwards, the continous phase is calculated and residuals are displayed for each field.

```
diagonal:  Solving for rho, Initial residual = 0, Final residual = 0, No Iterations 0
PIMPLE: iteration 1
smoothSolver:  Solving for Ux, Initial residual = 1, Final residual = 4.39809e-07, No Iterations 2
smoothSolver:  Solving for Uy, Initial residual = 1, Final residual = 4.98567e-07, No Iterations 2
smoothSolver:  Solving for Uz, Initial residual = 1, Final residual = 4.39908e-07, No Iterations 2
DILUPBiCGStab:  Solving for C7H16, Initial residual = 1, Final residual = 1.34756e-07, No Iterations 1
DILUPBiCGStab:  Solving for O2, Initial residual = 0.623435, Final residual = 8.4012e-08, No Iterations 1
DILUPBiCGStab:  Solving for CO2, Initial residual = 0, Final residual = 0, No Iterations 0
DILUPBiCGStab:  Solving for H2O, Initial residual = 0, Final residual = 0, No Iterations 0
DILUPBiCGStab:  Solving for h, Initial residual = 0.997314, Final residual = 1.34805e-07, No Iterations 1
T gas min/max   799.934, 800
GAMG:  Solving for p, Initial residual = 0.999998, Final residual = 0.0245563, No Iterations 2
diagonal:  Solving for rho, Initial residual = 0, Final residual = 0, No Iterations 0
time step continuity errors : sum local = 5.63895e-10, global = -5.19929e-10, cumulative = -5.19929e-10
rho min/max : 21.6899 21.6917
GAMG:  Solving for p, Initial residual = 0.0228333, Final residual = 9.09458e-07, No Iterations 7
diagonal:  Solving for rho, Initial residual = 0, Final residual = 0, No Iterations 0
time step continuity errors : sum local = 5.21535e-10, global = -5.2153e-10, cumulative = -1.04146e-09
rho min/max : 21.6899 21.6917
smoothSolver:  Solving for epsilon, Initial residual = 0.00241979, Final residual = 6.74282e-07, No Iterations 1
smoothSolver:  Solving for k, Initial residual = 1, Final residual = 3.62203e-07, No Iterations 2
ExecutionTime = 4.14 s  ClockTime = 4 s

Courant Number mean: 8.24166e-08 max: 2.30053e-06
deltaT = 3.61991e-06
Time = 6.56109e-06
```

To visualize the results, we open paraview by typing in the terminal

```
paraFoam -case $FOAM_RUN/aachenBomb/
```

and visualize the continuous fields.

This option will enable the user to visualize only the continuous fields characterizing the gaseous phase, such as velocity, temperature, pressure etc. In order to visualize the particles, we need to convert the results to `VTK` format by typing:

```
foamToVTK -case $FOAM_RUN/aachenBomb/
```

Open paraview, `paraFoam -case $FOAM_RUN/aachenBomb/`, load Lagrangian particle, by opening the following file `VTK/lagrangian/sprayCloud/sprayCloud.vtp.series`. After apply `Glyph` filter to it, set the `Glyph Type` to sphere and select `Scale Array` to `YC7H16(l)`.

Chapter 3

SprayFoam solver and Lagrangian libraries

3.1 sprayFoam solver

In this section we will have a look at how the parcels are created during the simulations and which libraries are related to the parcels; after we will change the breakup model to the PilchErdman breakup model.

The solver `sprayFoam` is implemented under the directory:

```
cd $FOAM_SOLVERS/lagrangian/sprayFoam
```

The following files are related to the sprayFoam slover:

```
.
|-- createClouds.H
|-- createFieldRefs.H
|-- createFields.H
|-- EEqn.H
|-- Make
|   |-- files
|   |-- options
|-- pEqn.H
|-- rhoEqn.H
|-- sprayFoam.C
|-- UEqn.H
|-- YEqn.H
```

Under the same folder you can find some other folders/files, which are related to different solvers with particles that are similar to sprayFoam. They are located under the same directory `sprayFoam` because they use some of the aforementioned files (check for example `engineFoam/Make/options`).

In the file `sprayFoam.C`, the following lines are related to the creation of the cloud, reading cloud properties, and updating the cloud properties during the simulation:

```
     . . .
38   #include "basicSprayCloud.H"
     . . .
58   #include "createFields.H"
     . . .
80   parcels.evolve();
     . . .
```

```
116  parcels.write();
     . . .
```

The first line, makes the class `basicSprayCloud` available to the main function. The second line creates the cloud object, specifically at the end of the file `createFields.H` (located in the same directory as `sprayFoam.C`), the following line `#include "createClouds.H"` denotes the creation of an object named `parcels` of type `basicSprayCloud`. The third line updates the parcels properties at the start of each iteration, properties like movement, phase change, collision, breakup, and the update of source terms for exchange of mass, momentum and energy in the NS equations. The last line `parcels.write();` writes the cloud data in our case directory.

When the solver is compiled, the class `basicSprayCloud.H` is made accessible to our main function, through the following lines in `Make/options` file:

```
EXE_INC = \
    . . .
     -I$(LIB_SRC)/lagrangian/basic/lnInclude \
     -I$(LIB_SRC)/lagrangian/intermediate/lnInclude \
     -I$(LIB_SRC)/lagrangian/spray/lnInclude \
     -I$(LIB_SRC)/lagrangian/distributionModels/lnInclude \
    . . .

EXE_LIBS = \
    . . .
     -llagrangian \
     -llagrangianIntermediate \
     -llagrangianTurbulence \
     -llagrangianSpray \
    . . .
```

The library `llagrangian` is related to the cloud object. The `llagrangianIntermediate` library is related to numerical schemes for solving the cloud movement, adding and solving the thermophysical properties of the cloud etc. The `llagrangianSpray` library contains all the sub-models of the cloud, such as collision models, radiation models, breakup models etc. Also the `distribustionModel` link, make available to the solver the some Probability Density Functions (PDF) that are used to set initial diameter of the parcels e.g. Rosin-Rammler PDF.

3.2 Change the secondary breakup model

In this section we present how one can change the secondary breakup model in our basic case `aachenBomb`. This can be done in two ways one can use the banana method or, read the code and understand it. The first way, is to modify the entry in a dictionary using "banana" word and allow the solver to give you hints on the available choices. For example, go to the tutorial directory and modify the breakupModel in the `sprayCloudProperties` dictionary:

```
cd $FOAM_RUN/aachenBomb
```

modify the line 164 of file `constant/sprayCloudProperties` to:

```
breakupModel    banana;
```

If you run the tutorial:

```
blockMesh
sprayFoam
```

you will get an error, and the solver will tell you that there is no such a breakup model of type "banana". Additionally, all the available breakup models will be listed:

```
--> FOAM FATAL ERROR:
Unknown breakupModel type banana

Valid breakupModel types :

7
(
ETAB
PilchErdman
ReitzDiwakar
ReitzKHRT
SHF
TAB
none
)
```

In this way one can change the secondary breakup model, only by reading the errors when one entry in your dictionary is incorrect. This method allows the user to change the breakup models without reading the code and is a good practice for new users to find out all the options that are available.

The second way, will allow the user to understand how the dictionary should look like by reading the code where the breakup model is implemented in OpenFOAM. Previously, we saw that the library `llagrangianSpray` contains the sub-models, also from the line `I$(LIB_SRC)/lagrangian/spray/lnInclude`, we automatically understand that this library is implemented in the following directory:

```
cd $FOAM_SRC/lagrangian/spray
```

The library `llagrangianSpray` contains the following directories; by typing the command `tree -L 2` we get the following output:

```
.
|-- clouds
|   |-- baseClasses
|   |-- derived
|   |-- Templates
|-- Make
|   |-- files
|   |-- options
|-- parcels
|   |-- derived
|   |-- include
|   |-- Templates
|-- submodels
    |-- AtomizationModel
    |-- BreakupModel
    |-- StochasticCollision
```

We observe that the breakup models are implemented under the directory `submodels/BreakupModel`. By typing the commands

```
cd submodels/BreakupModel && tree -L 1
```

we list all breakup models implemented in OpenFOAM:

```
.
|-- BreakupModel
|-- ETAB
|-- NoBreakup
```

```
|-- PilchErdman
|-- ReitzDiwakar
|-- ReitzKHRT
|-- SHF
|-- TAB
```

Suppose that we want to use the `PilchErdman` model as a secondary breakup model, therefore we go to the respective directory:

```
cd PilchErdman && ls
PilchErdman.C  PilchErdman.H
```

The `PilchErdman.H` contains the declaration of function and the `PilchErdman.C` contains the definition of the functions belonging to the class PilchErdman. This class has two private data

```
scalar B1_;
scalar B2_;
```

the following two constructors, one virtual constructor and one destructor:

```
//- Construct from dictionary
PilchErdman(const dictionary&, CloudType&);

//- Construct copy
PilchErdman(const PilchErdman<CloudType>& bum);

//- Construct and return a clone
virtual autoPtr<BreakupModel<CloudType>> clone() const
. . .

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

and one virtual member function, which updates the properties of the parcels with respect to secondary breakup.

```
//- Update the parcel properties
virtual bool update
(
    const scalar dt,
    const vector& g,
    scalar& d,
    scalar& tc,
    scalar& ms,
    scalar& nParticle,
    scalar& KHindex,
    scalar& y,
    scalar& yDot,
    const scalar d0,
    const scalar rho,
    const scalar mu,
    const scalar sigma,
    const vector& U,
    const scalar rhoc,
    const scalar muc,
    const vector& Urel,
    const scalar Urmag,
```

```
    const scalar tMom,
    scalar& dChild,
    scalar& massChild
);
```

From the above, the only scalars that need to be initialized are `B1_` and `B2_`. Also, in the definition of the constructor, once the submodel is created, the dictionary is looking for the values `B1` and `B2` unless default coefficient is set to true.

```
32  template<class CloudType>
33  Foam::PilchErdman<CloudType>::PilchErdman
34  (
35      const dictionary& dict,
36      CloudType& owner
37  )
38  :
39      BreakupModel<CloudType>(dict, owner, typeName),
40      B1_(0.375),
41      B2_(0.2274)
42  {
43      if (!this->defaultCoeffs(true))
44      {
45          this->coeffDict().readEntry("B1", B1_);
46          this->coeffDict().readEntry("B2", B2_);
47      }
48  }
```

Therefore, we can specify the submodel constants in the dictionary in two ways. First option is to set the value of `defaultCoeffs` to `true`, this is denoted by line 43. Second option to specify the model values `B1` and `B2` manually, lines 45-46.

Now we are ready to modify the dictionary, so as to change the breakup model:

```
cd $FOAM_RUN/aachenBomb
```

First, modify the value of the `breakupModel` to `PilchErdman` in the `constant/sprayCloudProperties` dictionary:

```
sed -i '/breakupModel/c\breakupModel PilchErdman;' constant/sprayCloudProperties
```

Secondly, add before the line `ReitzDiwakarCoeffs` one of the two ways to define the models coefficients:

```
PilchErdmanCoeffs
    {
        defaultCoeffs true;
    }
```

or

```
PilchErdmanCoeffs
    {
     B1 0.375;
     B2 0.2274;
    }
```

Finally, run the case:

```
foamListTime -rm
sprayFoam > log.sprayFoam &
```

If you check the the `log.sprayFoam`, you can see that PilchErdman secondary breakup model was selected.

Chapter 4

Model description

4.1 TAB model

The Taylor Analogy model, first suggested by [3], is based upon the analogy between a second-order harmonic oscillator, i.e. a forced Mass-Spring-Damper (MSD) system, and a fuel droplet that penetrates into a gaseous atmosphere. According to this analogy, the aerodynamic drag plays the role of the external force f which deforms the droplet, thereby initiating its oscillation. Further, surface tension acts as a restorative force that tries to maintain the sphericity of the droplet and to minimize its deformation. Therefore, in the MSD system, the surface tension plays the role of the spring force kx where x is the displacement of the droplet equator from its spherical (undisturbed) position. The viscous stresses due to the motion of the liquid inside the droplet are of a dissipative nature and play the role of the damping force $b\,\mathrm{d}x/\mathrm{d}t$. The second order differential equation of the MSD system is then formulated as:

$$m_\mathrm{d}\frac{\mathrm{d}^2x}{\mathrm{d}t^2} \;=\; -b\frac{\mathrm{d}x}{\mathrm{d}t} - kx + f\,, \tag{4.1}$$

where m_d is the mass of the droplet.

The TAB model keeps track only of the fundamental mode of oscillation, corresponding to the lowest order harmonic whose axis is aligned with the relative velocity vector between droplet and gas. This mode is dominant at small Weber numbers while for large Weber numbers other modes are contributing significantly to droplet breakup. The Weber number We is the ratio between inertial force add surface tension, $We = (\rho \boldsymbol{u}_\mathrm{rel}^2 r_\mathrm{d})/\sigma$.

In accordance with the Taylor analogy [3], the physical dependencies of the coefficients in the equation are the following.

$$\frac{f}{m_\mathrm{d}} \;=\; C_f\frac{\rho|\boldsymbol{u}_\mathrm{rel}|^2}{\rho_\mathrm{d} r_\mathrm{d}}\,, \qquad \frac{k}{m_\mathrm{d}} \;=\; C_k\frac{\sigma}{\rho_\mathrm{d} r_\mathrm{d}^3}\,, \qquad \frac{b}{m_\mathrm{d}} \;=\; C_b\frac{\mu_\mathrm{d}}{\rho_\mathrm{d} r_\mathrm{d}^2}\,, \tag{4.2}$$

where r_d is the radius of the droplet, σ is the gas-liquid surface tension, μ_d is the viscosity of the liquid fuel and, $C_\mathrm{f}\,,\ C_k$ and C_b are dimensionless coefficients.

Additionally, the displacement of the droplet is nondimensionalized according to $y = x/(C_r r)$ where C_r is a scaling dimensionless constant. By substituting these expressions, the equation of the harmonic oscillator (4.2) can be written as,

$$\frac{\mathrm{d}^2y}{\mathrm{d}t^2} \;=\; \frac{C_f}{C_r}\frac{\rho}{\rho_\mathrm{d}}\frac{|\boldsymbol{u}_\mathrm{rel}|^2}{r_\mathrm{d}^2} - C_k\frac{\sigma}{\rho_\mathrm{d} r_\mathrm{d}^3}y - C_b\frac{\rho_\mathrm{d}\mu_\mathrm{d}}{r_\mathrm{d}^2}\frac{\mathrm{d}y}{\mathrm{d}t}\,. \tag{4.3}$$

with breakup occurring if and only if $y > 1$. According to [3], for the constant relative speed, the solution of (4.3) is:

$$y(t) = \frac{C_f}{C_k C_r}We + e^{\frac{-t}{t_\mathrm{d}}}\left[\left(y_0 - \frac{C_f}{C_k C_r}We\right)cos(\omega t) + \frac{1}{\omega}\left(\dot{y}_0 + \frac{y_0 - \frac{C_f}{C_k C_r}We}{t_\mathrm{d}}\right)sin(\omega t)\right] \tag{4.4}$$

where:

$$y_0 = y(0)\,, \tag{4.5}$$

$$\dot{y}_0 = \frac{\mathrm{d}y}{\mathrm{d}t}(0)\,, \tag{4.6}$$

$$\frac{1}{t_\mathrm{d}} = \frac{C_b}{2}\frac{\mu_\mathrm{d}}{\rho_\mathrm{d} r_\mathrm{d}^2}\,, \tag{4.7}$$

and

$$\omega^2 = C_k \frac{\sigma}{\rho_\mathrm{d} r_\mathrm{d}^3} - \frac{1}{t_\mathrm{d}^2}\,. \tag{4.8}$$

The values of the constants C_f, C_k, C_b and C_r are calculated by employing a combination of experimental and theoretical results. More specifically, C_k and C_b are obtained by matching the fundamental oscillation frequency and the oscillation of the fundamental mode for the damping coefficient, and take the following values: $C_k = 8$ and $C_b = 5$ [1]. Also, in the TAB model its assumed that breakup occurs if and only if the amplitude of the oscillation of the north and south poles of the droplet become equal the droplet radius. From this assumption, and since $y = 1$ during breakup, we have that

$$C_r \;=\; \frac{1}{2}\,. \tag{4.9}$$

Further, according to experiments [2], the critical Weber number (We_cr) for breakup was found to be $We_\mathrm{cr} = 6$. According to [3], the model matches the experimental results if

$$\frac{C_k C_r}{C_f} \;=\; 2We_\mathrm{cr} = 12\,. \tag{4.10}$$

By inserting (4.9) into (4.10), we arrive at

$$C_f \;=\; \frac{1}{3}\,. \tag{4.11}$$

Regarding the atomization characteristics, the equation of the droplet size after breakup should be based on the energy conservation between the parent droplet and the product droplets by combining the droplet oscillation energy and surface energy.

We further note that, in reality, any ensemble of droplets will be polydisperse. The Sauter mean diameter r_{32} is defined as the droplet size in a monodisperse ensemble for which the total surface energy is equal to the total surface energy of an ensemble of polydisperse droplets when both ensembles have the same total area and total volume [4]. In the TAB model, the relation between the radius of the parent droplet r_d and the Sauter mean diameter r_{32} of the product droplets reads

$$\frac{r_\mathrm{d}}{r_{32}} \;=\; 1 + \frac{8K}{20} + \frac{\rho_\mathrm{d} r_\mathrm{d}^3}{\sigma}\left(\frac{\mathrm{d}y}{\mathrm{d}t}\right)^2\left(\frac{6K-5}{120}\right)\,. \tag{4.12}$$

In (4.12) the value of K is obtained via comparisons with experimentally measured droplet sizes, and is set to $K = \frac{10}{3}$. After breakup, the radius of the product droplets is chosen randomly from a χ^2 distribution, and the number of droplets can be predicted using the mass conservation constraint.

4.2 Modification of TAB model

In this section we propose a modification to the original TAB model. At high We, the aerodynamic force, f, plays a dominant role in the disintegration of the droplet. According to the original model, the coefficient C_f, which incorporates the effect of the external forces, is determined empirically.

Herein, we propose to calculate C_f dynamically for each droplet. More specifically, since f represents the aerodynamic drag, C_f can be cast as a function of the drag coefficient of the sphere C_D. In addition, due to the fact that the droplet diameters are small, we may approximate the

frontal area of a droplet by that of a perfect sphere with the same diameter. Therefore, we can write:

$$\frac{f}{m_{\rm d}} \;=\; \frac{\frac{1}{2}A_{\rm d}|\boldsymbol{u}_{\rm rel}|^2 C_{\rm D}\rho}{V_{\rm d}\rho_{\rm d}} \;=\; \frac{3}{4}C_{\rm D}\frac{|\boldsymbol{u}_{\rm rel}|^2\rho}{\rho_{\rm d}d_{\rm d}}\,, \tag{4.13}$$

where $A_{\rm d}$ is the frontal area of the droplet, $V_{\rm d}$ represent the volume of the droplet and $C_{\rm D}$ is defined as:

$$C_{\rm D} \;=\; \begin{cases} \frac{24}{Re_{\rm d}}(1+\frac{1}{6}Re_{\rm d}^{\frac{2}{3}}) & Re_{\rm d} < 1000\,, \\ 0.424 & Re_{\rm d} \geq 1000\,. \end{cases} \tag{4.14}$$

By combining (4.13) and the first relation in (4.2), we readily arrive in the following relation between $C_{\rm f}$ and $C_{\rm D}$,

$$C_{\rm f} \;=\; \frac{3}{8}C_{\rm D}\,. \tag{4.15}$$

Following the analysis of the TAB model, the remaining values of constants, $C_{\rm r}$ and $C_{\rm b}$ are kept as before, while the value of the $C_{\rm k}$ is calculated dynamically. Particularly, for each droplet, $C_{\rm f}$ is calculated using the relations (4.15) and (4.14), while $C_{\rm k}$ is calculated from (4.10). Also the diameters of the droplets after breakup are calculated as in the TAB model, namely, by employing the relation (4.12). According to the proposed modification, the parameters entering the equation for the harmonic oscillator (4.3) have different values for each droplet, based on the droplet characteristics and local flow conditions.

Chapter 5

Implementation in OpenFOAM

5.1 TAB in OpenFOAM

The TAB model is implemented in OpenFOAM under the following directory.

```
cd $FOAM_RUN/lagrangian/spray/submodels/BreakupModel/TAB/
```

As for the PilchErdman model, the declaration of the function is found in `TAB.H` and the definition in `TAB.C`. Additionally, two different functions are proposed for calculating the new droplet diameter after breakup, they are implemented in `TABSMDCalcMethod1.H` and `TABSMDCalcMethod2.H`. In the declaration file, `TAB.H`, one can observe the `TAB` class has as a public member the class `BreakupModel<CloudType>`. Furthermore, one public distinct type, `SMDMethod` of type enumeration, which will serve to select the method for determining the droplet diameter after breakup; and some private data.

```
63  template<class CloudType>
64  class TAB
65  :
66      public BreakupModel<CloudType>
67  {
68  public:
69
70      //- Enumeration for the SMD breakup calculation
71      enum SMDMethods
72      {
73          method1,
74          method2
75      };
76
77
78  private:
79
80      // Private data
81
82          // Inverse function approximation of the Rossin-Rammler Distribution
83          // used when calculating the droplet size after breakup
84          FixedList<scalar, 100> rrd_;
85
86
87          // Model constants
88
```

```
89              word SMDCalcMethod_;
90              SMDMethods SMDMethod_;
```

After, the run type information, two constructors (lines 102 and 105), one constructor for cloning the class (lines 108-114), one destructor (line 118) and the member functions `update` (lines 124-147) are declared:

```
93  public:
94
95      //- Runtime type information
96      TypeName("TAB");
97
98
99      // Constructors
100
101         //- Construct from dictionary
102         TAB(const dictionary& dict, CloudType& owner);
103
104         //- Construct copy
105         TAB(const TAB<CloudType>& im);
106
107         //- Construct and return a clone
108         virtual autoPtr<BreakupModel<CloudType>> clone() const
109         {
110             return autoPtr<BreakupModel<CloudType>>
111             (
112                 new TAB<CloudType>(*this)
113             );
114         }
115
116
117     //- Destructor
118     virtual ~TAB();
119
120
121     // Member Functions
122
123         //- Update the parcel diameter
124         virtual bool update
125         (
126             const scalar dt,
127             const vector& g,
128             scalar& d,
129             scalar& tc,
130             scalar& ms,
131             scalar& nParticle,
132             scalar& KHindex,
133             scalar& y,
134             scalar& yDot,
135             const scalar d0,
136             const scalar rho,
137             const scalar mu,
138             const scalar sigma,
139             const vector& U,
140             const scalar rhoc,
```

```
141                 const scalar muc,
142                 const vector& Urel,
143                 const scalar Urmag,
144                 const scalar tMom,
145                 scalar& dChild,
146                 scalar& massChild
147         );
148 };
```

The virtual function `update` checks if the droplet undergoes disintegration or not, and updates the parcels properties with respect to the secondary breakup. In what follows we present the class, along with explanatory comments:

```
87  template<class CloudType>
88  bool Foam::TAB<CloudType>::update
89  (
90      const scalar dt,   //Time step for solving the cloud
91      const vector& g,   //Local gravitational of other body-force acceleration
92      scalar& d,         //diameter
93      scalar& tc,        //Characteristic time (used in atomization and/or breakup model)
94      scalar& ms,        //Stripped parcel mass due to breakup
95      scalar& nParticle, //Number of Particle inside the Parcel
96      scalar& KHindex,   //Index for KH Breakup
97      scalar& y,         //Spherical deviation
98      scalar& yDot,      //Rate of change of spherical deviation
99      const scalar d0,   //Initial droplet diameter
100     const scalar rho,  //Parcel density
101     const scalar mu,   //Liquid dynamic viscosity
102     const scalar sigma,//Liquid surface tension
103     const vector& U,   //Velocity of the parcel
104     const scalar rhoc, //Density of continuous phase
105     const scalar muc,  //Viscosity of continuous phase
106     const vector& Urel,//Relative velocity
107     const scalar Urmag,//Magnitude of the relative velocity
108     const scalar tMom, //Momentum relaxation time
109     scalar& dChild,    //Characteristic diameter of the particle after disintegration
110     scalar& massChild  // see line 260 spray/parcels/Templates/SprayParcel/SprayParcel.C
111 )
```

The function starts with declaration of several variables, that will be used during the calculation of the breakup time and properties of the droplet after breakup. Also the magnitude of the oscillation is calculated.

```
112 {
113     Random& rndGen = this->owner().rndGen();
114
115     scalar r = 0.5*d; // define the radius of the droplet
116     scalar r2 = r*r;  // define the square of the radius
117     scalar r3 = r*r2; // define the radius at power of 3
118
119     scalar semiMass = nParticle*pow3(d);
120
121     // inverse of characteristic viscous damping time equation (4.7)
122     scalar rtd = 0.5*this->TABCmu_*mu/(rho*r2);
123
124     // oscillation frequency (squared)  equation (4.8)
125     scalar omega2 = this->TABComega_*sigma/(rho*r3) - rtd*rtd;
```

After that, if the distortion and oscillation are important, according to the breakup model, the droplet will undergo disintegration if and only if the magnitude of the undamped oscillation is greater then one.

```
127     if (omega2 > 0) // checks if the distortion and oscillation are important
128     {
129         scalar omega = sqrt(omega2);
```

```
130         scalar We = rhoc*sqr(Urmag)*r/sigma;
131         scalar Wetmp = We/this->TABtwoWeCrit_;
132
133         scalar y1 = y - Wetmp;
134         scalar y2 = yDot/omega;
135
136         scalar a = sqrt(y1*y1 + y2*y2);
137
138         // scotty we may have break-up
139         if (a+Wetmp > 1.0) //amplitude of undamped oscillation
```

Finally, if the breakup occurs the properties of the droplets inside the parcel after disintegration are updated.

```
140         {
141             scalar phic = y1/a;
142
143             // constrain phic within -1 to 1
144             phic = max(min(phic, 1), -1);
145
146             scalar phit = acos(phic);
147             scalar phi = phit;
148             scalar quad = -y2/a;
149             if (quad < 0)
150             {
151                 phi = constant::mathematical::twoPi - phit;
152             }
153
154             scalar tb = 0;
155
156             if (mag(y) < 1.0)
157             {
158                 scalar coste = 1.0;
159                 if ((Wetmp - a < -1) && (yDot < 0))
160                 {
161                     coste = -1.0;
162                 }
163
164                 scalar theta = acos((coste-Wetmp)/a);
165
166                 if (theta < phi)
167                 {
168                     if (constant::mathematical::twoPi - theta >= phi)
169                     {
170                         theta = -theta;
171                     }
172                     theta += constant::mathematical::twoPi;
173                 }
174                 tb = (theta-phi)/omega;
175
176                 // breakup occurs
177                 if (dt > tb)
178                 {
179                     y = 1.0;
180                     yDot = -a*omega*sin(omega*tb + phi);
181                 }
182
183             }
184
185             // update droplet size
186             if (dt > tb)
187             {
188                 scalar rs =
189                     r/(1.0 + (4.0/3.0)*sqr(y) + rho*r3/(8*sigma)*sqr(yDot)); //eq. 4.12 for y=1
190
191                 label n = 0;
192                 scalar rNew = 0.0;
193                 switch (SMDMethod_)
```

```
194                 {
195                     case method1:
196                         #include "TABSMDCalcMethod1.H"
197                         break;
198                     }
199                     case method2:
200                     {
201                         #include "TABSMDCalcMethod2.H"
202                         break;
203                     }
204                 }
205 
206                 if (rNew < r)
207                 {
208                     d = 2*rNew;
209                     y = 0;
210                     yDot = 0;
211                 }
212             }
213         }
214     }
215     else
216     {
217         // reset droplet distortion parameters
218         y = 0;
219         yDot = 0;
220     }
221 
222     // update the nParticle count to conserve mass
223     nParticle = semiMass/pow3(d);
224 
225     // Do not add child parcel
226     return false;
227 }
```

At this point, we know which function we should modify in case we want to implement the proposed secondary breakup. Therefore, we are ready to implement the above proposed breakup model.

5.2 Modified TAB

This section provides a step-by-step procedure on how to implement the above discussed model as a new library under the user directory. As the first step the Lagrangian library needs to be copied to the user directory.

```
cd $WM_PROJECT_DIR
cp -r --parents src/lagrangian/spray/ $WM_PROJECT_USER_DIR
```

Secondly, we need to modify the Make/files accordingly:

```
cd $WM_PROJECT_USER_DIR/src/lagrangian/spray
sed -i s/FOAM_LIBBIN/FOAM_USER_LIBBIN/ Make/files
sed -i s/liblagrangianSpray/libmylagrangianSpray/ Make/files
```

At this point we can compile the new library.

```
wmake
```

The new library, `libmylagrangianSpray`, is the same library as the initial one with a different name. The next step is to create the folder for the new model, for this purpose, we copy the TAB model and change the name to `myTAB` of the folder and the files respectively

```
cd $WM_PROJECT_USER_DIR/src/lagrangian/spray
cp -r submodels/BreakupModel/TAB/ submodels/BreakupModel/myTAB
mv submodels/BreakupModel/myTAB/TAB.C submodels/BreakupModel/myTAB/myTAB.C
mv submodels/BreakupModel/myTAB/TAB.H submodels/BreakupModel/myTAB/myTAB.H
```

Additionally, we need to substitute the string `TAB` by `myTAB`, except when its appears in the header file of `myTAB.H` (check it manually after running the following command), also delete the description in the header file.

```
sed -i s/TAB/myTAB/g myTAB.H
```

In order to compile the new submodel, we need to edit the file `parcels/include/makeSprayParcelBreakupModels.H`, adding the line:

```
#include "myTAB.H"
```

below

```
#include "SHF.H"
```

Also we need to add the following line, in the same file:

```
makeBreakupModelType(myTAB, CloudType);                                    \
```

below

```
makeBreakupModelType(ETAB, CloudType);                                     \
```

The differences between the original file and modified should look like this:

```
diff parcels/include/makeSprayParcelBreakupModels.H \
$FOAM_SRC/lagrangian/spray/parcels/include/makeSprayParcelBreakupModels.H

40d39
< #include "myTAB.H"
53d51
<  makeBreakupModelType(myTAB, CloudType);    \
```

Finally we can compile the library:

```
wmake
```

The next step is to modify the breakup model accordingly, specifically we need to modify the definition of the `update` function of `myTAB.C`. For this purpose go to the `myTAB` folder

```
cd $WM_PROJECT_USER_DIR/src/lagrangian/spray/submodels/BreakupModel/myTAB/
```

and we make the following modification. We add the following model constants as private data in the `myTAB.H` file:

```
    // Model constants
    scalar y0_;
    scalar yDot0_;
    scalar Comega_;
    scalar Cmu_;
    scalar WeCrit_;
    word SMDCalcMethod_;
    SMDMethods SMDMethod_;
```

We change the entire definition of first constructor in `myTAB.C` by replacing the following lines:

```
template<class CloudType>
Foam::TAB<CloudType>::TAB
(
    const dictionary& dict,
    CloudType& owner
```

```
)
:
    BreakupModel<CloudType>(dict, owner, typeName, true),
    SMDCalcMethod_(this->coeffDict().lookup("SMDCalculationMethod"))
{
```

with:

```
template<class CloudType>
Foam::myTAB<CloudType>::myTAB
(
    const dictionary& dict,
    CloudType& owner
)
:
    BreakupModel<CloudType>(dict, owner, typeName),
    y0_(0),
    yDot0_(0),
    Comega_(8),
    Cmu_(5),
    WeCrit_(12),
    SMDCalcMethod_(word("method1"))
{
    if (!this->defaultCoeffs(true))
    {
        this->coeffDict().readEntry("y0", y0_);
        this->coeffDict().readEntry("yDot0", yDot0_);
        this->coeffDict().readEntry("Comega", Comega_);
        this->coeffDict().readEntry("Cmu", Cmu_);
        this->coeffDict().readEntry("WeCrit",WeCrit_);
        SMDCalcMethod_(word(this->coeffDict().lookup("SMDCalculationMethod")));
    }
```

Also we modify the second constructor in `myTAB.C` by replacing the following lines:

```
template<class CloudType>
Foam::TAB<CloudType>::TAB(const TAB<CloudType>& bum)
:
    BreakupModel<CloudType>(bum),
    SMDCalcMethod_(bum.SMDCalcMethod_)
{}
```

with

```
template<class CloudType>
Foam::myTAB<CloudType>::myTAB(const myTAB<CloudType>& bum)
:
    BreakupModel<CloudType>(bum),
    y0_(bum.y0_),
    yDot0_(bum.yDot0_),
    Comega_(bum.Comega_),
    Cmu_(bum.Cmu_),
    WeCrit_(bum.WeCrit_),
    SMDCalcMethod_(bum.SMDCalcMethod_)
{}
```

Finally we change the function update in the file `myTAB.C` to:

```
template<class CloudType>
bool Foam::myTAB<CloudType>::update
(
    const scalar dt,
    const vector& g,
    scalar& d,
    scalar& tc,
    scalar& ms,
    scalar& nParticle,
    scalar& KHindex,
    scalar& y,
    scalar& yDot,
    const scalar d0,
    const scalar rho,
    const scalar mu,
    const scalar sigma,
    const vector& U,
    const scalar rhoc,
    const scalar muc,
    const vector& Urel,
    const scalar Urmag,
    const scalar tMom,
    scalar& dChild,
    scalar& massChild
)
{
    Random& rndGen = this->owner().rndGen();

    scalar r = 0.5*d;
    scalar r2 = r*r;
    scalar r3 = r*r2;

    scalar semiMass = nParticle*pow3(d);


// calculating Weber number
scalar We = rho*sqr(Urmag)*d/sigma;

if(We>5)
{
//Reynolds number of the droplet
scalar Red =(rhoc*Urmag*d)/muc;
//Make sure that is greater then the machine epsilon
Red=max(SMALL,Red);
//
scalar CD=0.424;
//Reynolds number of droplets is less than 1000 the drag coefficient need to be updated
if (Red<=1000.0)
CD=24.0/Red*(1+1.0/6.0*pow(Red,0.66666));
        scalar Cf=3.0/8.0*CD;
        scalar Cb=0.5;
        scalar Comega_=12.0*Cf/Cb;
```

```
    // inverse of characteristic viscous damping time
    scalar rtd = 0.5*Cmu_*mu/(rho*r2);

// oscillation frequency (squared)
    scalar omega2 = Comega_*sigma/(rho*r3) - rtd*rtd;

    if (omega2 > 0)
    {
        scalar omega = sqrt(omega2);
        scalar We = rhoc*sqr(Urmag)*r/sigma;
        scalar Wetmp = We/WeCrit_;

        scalar y1 = y - Wetmp;
        scalar y2 = yDot/omega;

        scalar a = sqrt(y1*y1 + y2*y2);

        // scotty we may have break-up
        if (a+Wetmp > 1.0)
        {
            scalar phic = y1/a;

            // constrain phic within -1 to 1
            phic = max(min(phic, 1), -1);

            scalar phit = acos(phic);
            scalar phi = phit;
            scalar quad = -y2/a;
            if (quad < 0)
            {
                phi = constant::mathematical::twoPi - phit;
            }

            scalar tb = 0;

            if (mag(y) < 1.0)
            {
                scalar coste = 1.0;
                if ((Wetmp - a < -1) && (yDot < 0))
                {
                    coste = -1.0;
                }

                scalar theta = acos((coste-Wetmp)/a);

                if (theta < phi)
                {
                    if (constant::mathematical::twoPi - theta >= phi)
                    {
                        theta = -theta;
                    }
                    theta += constant::mathematical::twoPi;
                }
                tb = (theta-phi)/omega;
```

```
                // breakup occurs
                if (dt > tb)
                {
                    y = 1.0;
                    yDot = -a*omega*sin(omega*tb + phi);
                }

            }

            // update droplet size
            if (dt > tb)
            {
                scalar rs =
                    r/(1.0 + (4.0/3.0)*sqr(y) + rho*r3/(8*sigma)*sqr(yDot));

                label n = 0;
                scalar rNew = 0.0;
                switch (SMDMethod_)
                {
                    case method1:
                    {
                        #include "TABSMDCalcMethod1.H"
                        break;
                    }
                    case method2:
                    {
                        #include "TABSMDCalcMethod2.H"
                        break;
                    }
                }

                if (rNew < r)
                {
                    d = 2*rNew;
                    y = 0;
                    yDot = 0;
                }
            }
        }
    }
}
    else
    {
        // reset droplet distortion parameters
        y = 0;
        yDot = 0;
    }

    // update the nParticle count to conserve mass
    nParticle = semiMass/pow3(d);

    // Do not add child parcel
    return false;
```

```
}
```

At this moment we are ready to compile the library.

```
cd $WM_PROJECT_USER_DIR/src/lagrangian/spray
wmake
```

5.3 Compiling the solver using the new library

The next step is to add the newly-implemented Lagrangian library to our solver. For this reason we will change the existing solver `sprayFoam`, add the newly implemented Lagrangian library and compile it under a new name `mySprayFoam`.

The first step is to copy the `sprayFoam` solver's folder to the user directory, to rename the folder to `mySprayFoam` and to delete the unnecessary folders/files:

```
cd $WM_PROJECT_DIR
cp -r --parents applications/solvers/lagrangian/sprayFoam/ $WM_PROJECT_USER_DIR
cd $WM_PROJECT_USER_DIR/applications/solvers/lagrangian/
mv sprayFoam mySprayFoam && cd mySprayFoam
mv sprayFoam.C mySprayFoam.C
rm -r engineFoam simpleSprayFoam  sprayDyMFoam
```

Secondly, one should add the new lagrangian library and change the compilation options accordingly:

```
sed -i s/sprayFoam/mySprayFoam/g Make/files
sed -i s/FOAM_APPBIN/FOAM_USER_APPBIN/ Make/files
sed -i 's/(LIB_SRC)\/lagrangian\/spray\/lnInclude/(WM_PROJECT_USER_DIR)\/src\/lagrangian\/spray\/lnInclude/' Make/options
sed -i 's/llagrangianSpray/lmylagrangianSpray/' Make/options
sed -i '/^EXE_LIBS =.*/a -L\$(FOAM_USER_LIBBIN) \\' Make/options
```

Finally, the new solver `mySprayFoam` can be compiled by running the following command:

```
wmake
```

The newly implemented model can now be used. The next chapter will discuss in more detail how to do so.

5.4 Running the case

This chapter shows how to use the newly implemented model and how its respective dictionary should look like. For this purpose we will modify the existing tutorial `aachenBomb` that we have described in second chapter. First, we need to modify the `sprayProperties` dictionary, specifically we need change the breakup model that is read from `constant/sprayProperties`, therefore first we need to go to the run directory:

```
cd $WM_PROJECT_USER_DIR/run/aachenBomb/
```

and change the entry for the breakup model in the `constant/sprayCloudProperties` dictionary from `TAB` to `myTAB`

```
sed -i 's/breakupModel TAB;/breakupModel myTAB;/' constant/sprayCloudProperties
```

also, also we need to specify the model constants by adding the following lines before `ReitzDiwakarCoeffs`:

```
myTABCoeffs
{
    solveOscillationEq no;
    y0              0;
    yDot0           0;
```

```
    Cmu             10;
    Comega          8;
    WeCrit          12;
    SMDCalculationMethod  method1;
}
```

The next step is to generate the mesh:

```
blockMesh
```

and finally run the case, using `mySprayFoam` solver:

```
mySprayFoam >log.mySprayFoam&
```

One can read the `log.mySprayFoam` file to ensure that the `myTAB` model has been selected as a breakup model. In order to visualize the results, as before, we need to convert the results to VTK format and use the paraview:

```
foamToVTK
paraFoam
```

The above mention model, have been closely studied and assessed its capability for spray flows, where disintegration plays an important role. For a detailed elaboration, and comparison against experimental results, the reader is advised to check [5].

Study questions

1. Use the new library directly to your tutorial. Why doesn't the solver recognize the newly implemented breakup model?
2. Adjust the tutorial case so as the ETAB model is selected as a secondary breakup.
3. List all the distribution function available in openFoam for the initial droplet diameters.
4. List all the evaporation models available in openFoam.
5. Modify the sprayCloudProperties dictionary so as to select an atomization model.

Bibliography

[1] H. Lamb. *Hydrodynamics*. Dover, 6th edition edition, 1945.

[2] J. A. Nicholls. Stream and droplet breakup by shock waves. *NASA-SP-194 Technical Report*, pages 126–128, 1972.

[3] P. J. O'Rourke and A. A. Amsden. The tAB method for numerical calculation of spray droplet breakup. *SAE Technical Paper Series, 872089*, 1987.

[4] W. A. Sirignano. *Fluid Dynamics and Transport of Droplets and Sprays*. Cambridge University Press, 2 edition, 2010.

[5] C. Sula, H. Grosshans, and M. Papalexandris. Assessment of droplet breakup models for spray flow simulations. *Flow, Turbulence and Combustion*, 2020.