Cite as: Adib, M.: Implementation of a Monodisperse Population Balance Model in laminar combustion model. In Proceedings of CFD with OpenSource Software, 2022, Edited by Nilsson. H.,
http://dx.doi.org/10.17196/OS_CFD#YEAR_2022

CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

Implementation of a Monodisperse Population Balance Model in laminar combustion model

Developed for OpenFOAM-v2006

Author:
Mo Adib
Carleton University
mo.adib@carleton.ca

Peer reviewed by:
Dr. Reza Kholghy
Leandro Lucchese
Saeed Salehi

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.

January 26, 2023

Learning outcomes

The main requirements of a tutorial in the course is that it should teach the four points: How to use it, The theory of it, How it is implemented, and How to modify it. Therefore the list of learning outcomes is organized with those headers.

The reader will learn:

How to use it:

- how to use `laminarSoot` combustion model with `reactingFoam`
- how to set up cases that can be used with `laminarSoot` to simulate soot formation in combustion or pyrolysis of hydrocarbons

The theory of it:

- how the proposed monodisperse population balance model (MPBM) describes soot evolution
- how the soot model impacts the production/destruction rate of species

How it is implemented:

- how a combustion model communicates with `reactingFoam` solver
- how to calculate soot inception, growth, oxidation and coagulation rates within `laminarSoot` library and utilize them to solve transport equations of tracked fields

How to modify it:

- how the reaction rates and heat release obtained from gas chemistry should be modified to account for the production/consumption of species mass and energy by soot inception, growth and oxidation

Prerequisites

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

- Fundamental knowledge of fluid mechanics, combustion, chemistry, and soot structure and composition.
- Familiarity with computational fluid dynamics (CFD) and transport equation of physical properties
- A good knowledge of structure of solvers and libraries of OpenFOAM [1, 2]
- Understanding C++ syntax and accessing the object properties and methods
- How to create, modify and run OpenFOAM cases [3, 4]

[1] https://www.openfoam.com/documentation/guides/v2112/doc/
[2] http://www.tfd.chalmers.se/ hani/kurser/OS_CFD/
[3] https://www.openfoam.com/documentation/tutorial-guide/1-introduction
[4] https://www.openfoam.com/documentation/user-guide

Contents

Nomenclature

Acronyms

CFD	Computational Fluid Dynamics
DEM	Discrete Element Modelling
HACA	Hydrogen abstraction carbon addition
MPBM	Monodisperse Population Balance Model
SPBM	Sectional Population Balance Model

English symbols

Av	Avogadro's number	$6.02214076 \times 10^{23}$ 1/mol
C_{tot}	Carbon content of soot particles	mol/kg
d	diameter	m
d_g	Gyration diameter	m
d_m	Mobility diameter	m
d_p	Primary particle diameter	m
H_{tot}	Hydrogen content of soot particles	mol/kg
k	reaction rate constant	$\mathrm{m}^3/\mathrm{mol} - \mathrm{s}$
k_B	Boltzmann constant	$1.3806488 \times 10^{-23}$ $\mathrm{m}^2\mathrm{kg}/\mathrm{s}^2{-}\mathrm{K}$
m	mass	kg
n_p	Number of primary particles	
N_{agg}	Number density of agglomerates	mol/kg
N_{pri}	Number density of primary particles	mol/kg
T	Temperature	K
W	Molecular weight	kg

Greek symbols

β	Collision frequency	m^3/s
λ	Mean free path	m
ν	Gas kinematic viscosity	m^2/s
ω	The rate of production/consumption of species	$\mathrm{mol}/\mathrm{m}^3 - s$
ρ	Gas density	kg/m^3
ρ_{soot}	Soot density	kg/m^3

Subscripts

ad	Adsorption
agg	Agglomerate
coag	Coagulation
cont	Continuum
fm	Free molecular
fw	forward
FWD	forward
g	Gyration

grow	Surface growth
inc	Inception
m	Mobility
ox	Oxidation
pri	Primary particle
rc	reactive
REAC	reactive
REV	reverse
rv	reverse

Chapter 1

Introduction

1.1 Background

Soot (black carbon) is a major air pollutant, and the third strongest contributor to climate change [1]. Carbon black (CB) with similar synthesis process and properties to soot is the largest industrially produced nanomaterial by value and volume (15 megatons per year with a value of $17B) with applications as a reinforcing agent in rubber [2] and conductive additive in lithium-ion batteries [3]. Controlling properties of these particles is essential to determine functional properties of CB and environmental effects of soot, but it is challenging due to the complexity of soot formation, its coupling with gas phase chemistry [4], and fluid dynamic effects. So, accurate predictive models are required to describe fluid flow, gas chemistry and different steps of soot formation and evolution in a coupled manner.

Soot morphology can be described accurately by mesoscale simulations, such as Discrete Element Modeling (DEM) [5], but they are computationally expensive and difficult to interface with the computational fluid dynamic (CFD) methods. So, sectional population balance models (SPBM) coupled with power-laws are used in flow reactors [6] and laminar flames [7]. However, their computational cost grows exponentially by increasing the number of sections.

Monodisperse population balance models (MPBM) are alternative modelling approaches that describe average particle properties by tracking their total concentration, mass and area [8], but their accuracy depends on the assumptions about particle morphology (e.g. approximating agglomerates as monodisperse and perfect spheres). However, when inception and surface growth are short [9] and high particle (number) concentrations are formed [10], they lead to rapid attainment of self-preserving size distributions (SPSD) and agglomerates having asymptotic structure [11]. In these conditions, a MPBM can achieve accuracies on par with DEM [5], SPBM [12] and experimental data [8].

Due to high computational costs of the detailed description of gas chemistry, pseudo one-dimensional approaches are often used with detailed reaction mechanisms for prediction of species and soot in laminar flames and flow reactors [13]. However, one dimensional assumption is not valid when probes disturb the fluid flow or gravity effects are significant in flames [14] or when recirculation occurs in flow reactors. They may alter the chemistry of the gas and thereby change the soot formation. Capturing these effect requires a soot model coupled with chemistry solver and fluid dynamics. So, a combustion model that links reacting mixture solver to chemistry can provide an effective framework for the implementation of soot model.

1.2 Laminar combustion model

The main role of a combustion model in `reactingFoam` is to calculate (i) the net production rate of species contributes that appears as a source term in the transport equation of mass fraction `Y` and (ii) the net heat release from reaction that acts as a source term in conservation of enthalpy

`he` (this will be further investigated in Sec. 3.8). A full list of available combustion models in OpenFOAM can be found in the source directory under `$FOAM_SRC/combustionModels`. However, only `laminar` and `EDC` model are used in tutorial examples. These two models differ in how they treat the net production rate, `R`, and heat release `Qdot`. `laminar` model uses Arrhenius form expressions to calculate reaction rates neglecting the effect of turbulent fluctuations on reaction rates, which proves accurate for laminar flames, while `EDC` incorporates the turbulence quantities in calculation of reaction rate and heat release.

1.3 Objectives

This project aims to extend `laminar` combustion model to develop a new library, `laminarSoot`, which incorporates a MPBM for modelling soot formation by

- adding four additional transport equations to track the number density of primary particles (N_{pri}) and agglomerates (N_{agg}), total carbon (C_{tot}) and hydrogen (H_{tot}) content of soot particles
- implementing new functionalities to calculate contribution of inception, growth, coagulation and oxidation of soot to source terms of each transport equation
- modifying the production rates and rate of heat release to satisfy the conservation of species mass and energy.

Finally, `laminarSoot` is utilized with `reactingFoam` in a simple test case to test the performance of new library and evaluate the results.

Chapter 2

Theory

In this chapter, an overview of mathematical basic of the soot model is presented, which includes the transport equations, characterization of soot morphology and calculation of source terms corresponding to each step of soot evolution. All equations will be used to implement the soot model in `laminarSoot` library.

2.1 Transport equations

MPBM relies on the Eulerian description of particles where certain physical properties representative of particle population such as number density, mass or surface area are treated as continuous quantities that can be described by solving scalar transport equations. The formulation of soot model used in this project follows MPBM proposed by Kholghy and Kelesidis [8] based on number density of agglomerates, total carbon content and total surface area of particles. Here, we modify the original model by tracking the number density of primary particles ($\mathrm{N_{pri}}$) and agglomerates ($\mathrm{N_{agg}}$), total carbon ($\mathrm{C_{tot}}$) and hydrogen ($\mathrm{H_{tot}}$) content of soot particles by solving the following transport equations

$$\frac{\partial}{\partial t}(\rho N_{agg}) + \nabla \cdot (\rho u N_{agg}) + \nabla^2(\rho D N_{agg}) = \rho \left(S^N_{inc} + S^N_{coag}\right), \tag{2.1}$$

$$\frac{\partial}{\partial t}(\rho N_{pri}) + \nabla \cdot (\rho u N_{pri}) + \nabla^2(\rho D N_{pri}) = \rho S^N_{inc}, \tag{2.2}$$

$$\frac{\partial}{\partial t}(\rho C_{tot}) + \nabla \cdot (\rho u C_{tot}) + \nabla^2(\rho D C_{tot}) = \rho \left(S^C_{inc} + S^C_{grow} + S^C_{ox}\right), \tag{2.3}$$

$$\frac{\partial}{\partial t}(\rho H_{tot}) + \nabla \cdot (\rho u H_{tot}) + \nabla^2(\rho D H_{tot}) = \rho \left(S^H_{inc} + S^H_{grow} + S^H_{ox}\right). \tag{2.4}$$

In Equations (2.1)-(2.4), S represents the source terms where superscripts indicates the transport equation and subscripts denotes the process the source term is associated with. For example, $\mathrm{S^C_{grow}}$ accounts for the addition of carbon to soot particles via surface growth.

2.2 Morphology

Soot particles are formed as agglomerates of spherical primary particles. Incipient soot has spherical shape, and collision and attachment of these spheres results in fractal-like agglomerate. Hereafter, the word *"particle"* refers to soot both in spherical and agglomerate shape. The morphology of soot agglomerates are characterized by primary particle, mobility, and gyration diameters. These diameters can be related to each other by number of primary particles using power-laws. Mobility and gyration diameters are the diameter of the sphere with the same translational and rotational

properties of an agglomerate, respectively. Figure 2.1 illustrates the schematics of sample soot agglomerates with 12 primary particles and depicted d_p, d_m, d_g.

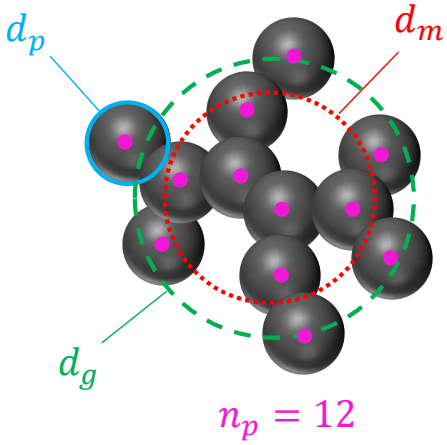

Figure 2.1: The schematics of a soot agglomerates with 12 primary particles ($n_p = 12$). Primary particle (d_p), mobility (d_m), and gyration (d_g) are shown.

n_p is the number of primary particles in each agglomerate that can be obtained by dividing the number density of primary particles by that of agglomerates,

$$n_p = \frac{N_{pri}}{N_{agg}}. \tag{2.5}$$

Primary particle diameter, d_p, can be obtained from total carbon and number density of primary particles using

$$d_p = \left(\frac{6}{\pi} \frac{C_{tot} \cdot W_{carbon}}{\rho_{soot}} \frac{1}{N_{pri} \cdot Av} \right)^{1/3}. \tag{2.6}$$

The DEM-derived power-laws relate d_m and d_g to d_p and n_p as

$$d_m = d_p \cdot n_p^{0.45}, \tag{2.7}$$

$$d_g = \begin{cases} d_m/(n_p^{-0.2} + 0.4), & \text{if } n_p > 1.5 \\ d_m/1.29. & \text{if } n_p \leq 1.5 \end{cases} \tag{2.8}$$

d_m and d_g are used to calculate the collision and attachment rate of soot agglomerates that accounts for the coagulation source term.

2.3 Diffusion of soot particles

The diffusion coefficient of soot particle, D, used in Equations (2.1)-(2.4) is calculated as

$$D = \frac{k_B T}{f}, \tag{2.9}$$

where f is the friction factor of particles in gas, and it is calculated for free molecular to continuum regimes as

$$f = \frac{3\pi\mu d_m}{C}, \tag{2.10}$$

$$C = 1 + \frac{2\lambda}{d_m}\left(1.21 + 0.4exp(\frac{-0.78d_m}{\lambda})\right), \tag{2.11}$$

where λ is the mean free path of gas given as:

$$\lambda = \frac{\mu}{\rho}\sqrt{\frac{\pi W_{gas}}{2k_B AvT}}. \tag{2.12}$$

2.4 Coagulation

Coagulation is the process during which solid and hard soot particles collide and attach at point of contact leading to larger agglomerates. This process conserves the soot mass and composition and number density of primary particles, so coagulation only affects N_{agg} (Equation (2.1)). S^{N}_{coag} accounts for the decay rate of N_{agg} by the binary collision of soot particles by

$$S^{N}_{coag} = -\frac{1}{2}\beta N^2_{agg}, \tag{2.13}$$

where β is harmonic mean of collision frequency in the continuum (β_{cont}) and free molecular (β_{fm}) regimes enhanced by %82 accounting for the polydispersity of particles [15] using the following relations

$$\beta = 1.82\frac{\beta_{fm}\beta_{cont}}{\beta_{fm} + \beta_{cont}}, \tag{2.14}$$

$$\beta_{fm} = 4\sqrt{\frac{\pi k_b T}{m_{agg}}}d_g^2, \tag{2.15}$$

$$\beta_{cont} = 8\frac{k_b T}{3\mu}\left(1 + \frac{2\lambda}{d_m}\left[1.21 + 0.4e^{-\frac{0.78d_m}{\lambda}}\right]\right). \tag{2.16}$$

2.5 Surface growth via HACA

Hydrogen abstraction carbon addition (HACA) is a major pathway for soot mass growth where active reaction sites on particles form bonds with acetylene molecule (C_2H_2). HACA mechanism [16] is described by a set of elementary with a given rates that are listed in Table 2.1. The HACA rate is defined as the absolute rate change of concentration of C_2H_2 ($\omega_{c_2h_2}$) via HACA mechanism as

$$\omega_{c_2h_2} = \alpha k_{f4}[\mathrm{C_2H_2}][\mathrm{C_{soot^\circ}}], \tag{2.17}$$

$$\frac{d}{dt}[\mathrm{C_2H_2}] = -\omega_{c_2h_2}. \tag{2.18}$$

In Equation (2.17), k_{f4} refers to forawrd reaction rate constant of 4th reaction in reaction 4 in Table 2.1. The contribution of HACA to growth source terms can be computed from HACA rate considering the number of carbon atoms in C_2H_2 and number of arm-chair and zig-zag hydrogenated sites on soot particle [17] using

$$S^{C}_{grow|HACA} = 2\omega_{c_2h_2}/\rho, \tag{2.19}$$

$$S^{H}_{grow|HACA} = 0.25\omega_{c_2h_2}/\rho. \tag{2.20}$$

In Equation (2.17), α is the surface reactivity of soot defined by an empirical relation [16] as

$$\alpha = \tanh\left(\frac{12.56 - 0.00563\cdot T}{\log_{10}\left(\frac{\rho_{soot}\frac{\pi}{6}d_p^3\cdot Av}{W_{carbon}}\right)} - 1.38 + 0.00068\cdot T\right). \tag{2.21}$$

Table 2.1: Rate coefficients for the various surface reactions in Arrhenius form $k = AT^n \cdot e^{-E/RT}$

No.	Reaction		A $\left[\frac{m^3}{mol \cdot s}\right]$	n	$\frac{E}{R}$[K]
1	$C_{soot-H} + H \rightleftarrows C_{soot^\circ} + H_2$	f	4.17×10^7	0	6542.52
		r	3.9×10^6	0	5535.98
2	$C_{soot-H} + OH \rightleftarrows C_{soot^\circ} + H_2O$	f	10^4	0.734	719.68
		r	3.68×10^2	1.139	8605.94
3	$C_{soot^\circ} + H \longrightarrow C_{soot} + H_2O$	f	10^4	0.734	719.68
4	$C_{soot^\circ} + C_2H_2 \longrightarrow C_{soot-H}$	f	80	1.56	1912.43
5	$C_{soot^\circ} + O_2 \longrightarrow 2\,CO$	f	2.2×10^6	0	3774.53
6	$C_{soot}-H + OH \longrightarrow CO + \frac{1}{2} H_2$	f	0.13	0	0

$[C_{soot^\circ}]$ is the concentration of dehydrogenated site on soot particle computed by

$$[C_{soot^\circ}] = A_{tot} \frac{\rho}{Av} \chi_{soot^\circ}. \tag{2.22}$$

A_{tot} is the total surface area of soot particles obtained as

$$A_{tot} = N_{pri} Av \cdot \pi d_p^2, \tag{2.23}$$

χ_{soot° is the number of active reaction sites per unit surface area of particles.

$$\chi_{soot^\circ} = \frac{k_{f1}[H] + k_{f2}[OH]}{k_{r1}[H_2] + k_{r2}[H_2O] + k_{f3}[H] + k_{f4}[C_2H_2] + k_{f5}[O_2] + k_{f1}[H] + k_{f2}[OH]} \chi_{soot_{CH}}, \tag{2.24}$$

where $\chi_{soot_{CH}} = 2.3 \times 10^{19} m^{-2}$. In Equation (2.24), k_{r1} denotes the reverse rate of the first reaction in Table 2.1, and the rest of reaction rates follow the same naming convention.

2.6 Oxidation via HACA

The carbon atoms on the surface of soot are oxidized via reaction with O_2 molecules and OH radicals which decreases total carbon of soot and releases CO and H_2 molecules to gas mixture. The oxidation process is described by HACA mechanism. Here, we assume that oxidation does not change the hydrogen content of soot particles. The absolute rate change of O_2 molecules (ω_{o2}) and OH radicals (ω_{oh}) by oxidation is calculated as

$$\omega_{o2} = \alpha k_{f5} [O_2][C_{soot^\circ}], \tag{2.25}$$

$$\omega_{oh} = \alpha k_{f6} [OH] N_{agg} \rho. \tag{2.26}$$

The oxidation source term is calculated considering the number of carbon atoms removed from soot through each oxidation pathway by

$$S_{ox}^C = -(2\omega_{o2} + \omega_{oh})/\rho, \tag{2.27}$$

2.7 Inception

The inception is described using reactive dimerization of polycylic aromatic hydrocarbons (PAHs) [18] where collision of two PAH molecules form physically-bonded dimers followed by their carbonization that results in new soot particles. This two step process can be described as

$$PAH_i + PAH_j \underset{k_{REV}}{\overset{k_{FWD}}{\rightleftarrows}} Dimer_{ij}^*, \tag{2.28}$$

$$\mathrm{Dimer}^{*}_{\mathrm{ij}} \xrightarrow{\mathrm{k_{REAC}}} \mathrm{Dimer_{ij}}. \tag{2.29}$$

In Equation (2.28), k_{FWD} is the forward rate of physical dimerization and computed as

$$k_{FWD} = 2.2 \cdot 0.1 \cdot Av \cdot d_{ij}^2 \sqrt{\frac{8\pi k_B T}{m_{ij}}}, \tag{2.30}$$

where $d_{ij} = 2d_i d_j/(d_i + d_j)$ and $m_{ij} = m_i m_j/(m_i + m_j)$ are reduced diameter and mass of PAH molecules in the dimer, respectively. The mass of PAH is calculated by dividing the molecular weight by Avogadro's number. The diameter is estimated by assuming a sphere with the mass of one PAH molecule and an estimated density [19] using

$$m_{PAH} = \frac{W_{PAH}}{Av}, \tag{2.31}$$

$$\rho_{PAH} = 171943.5197 \frac{n_{C,PAH} W_{carbon} + n_{H,PAH} W_{hydrogen}}{n_{C,PAH} + n_{H,PAH}}, \tag{2.32}$$

$$V_{PAH} = \frac{m_{PAH}}{\rho_{PAH}}, \tag{2.33}$$

$$d_{PAH} = \left(\frac{6V_{PAH}}{\pi}\right)^{1/3}. \tag{2.34}$$

The reverse rate of physical dimerization, $\mathrm{k_{REV}}$, is calculated from $\mathrm{k_{FWD}}$ and equilibrium coefficient of physical dimerization as

$$k_{REV} = k_{FWD} 10^{-b} e^{-a\epsilon \ln(10)/(RT)}, \tag{2.35}$$

$$\epsilon = cW_{ij} - d, \tag{2.36}$$

where $W_{ij} = W_i W_j/(W_i + W_j)$ is the reduced molecular mass of dimer, a = 0.115 (obtained from pyrere dimerization data [20]) and b=1.8 [18], c=933420 j/kg, and d=34053 j/mol [18].

The rate of chemical bond formation, $\mathrm{k_{REAC}}$ is defined in the Arrhenius form [6] as

$$k_{REAC} = 5 \times 10^6 \cdot e^{(-96232/RT)}. \tag{2.37}$$

Assuming a steady state condition for the physical dimers, $\partial[\mathrm{Dimer^{*}_{ij}}]/\partial \mathrm{t} = 0$, the formation of dimer can be obtained as

$$\omega_{dimer_{ij}} = k_{REAC} \frac{k_{FWD}[PAH_i][PAH_j]}{k_{REV} + k_{REAC}}. \tag{2.38}$$

The PAHs forming the dimer is removed from the gas mixture due to inception at the same rate as dimerization meaning that

$$\omega_{PAH_i} = \omega_{PAH_j} = -\omega_{dimer_{ij}}. \tag{2.39}$$

Here, we assume the smallest soot particle are 2 nm in diameter corresponding to a spherical particle with 378 carbon and 20 hydrogen atoms, respectively. The contribution of each dimer to number density (both agglomerates and primary particles), carbon and hydrogen content of particles is proportional to the mass, number of carbon and hydrogen atoms of that dimer, respectively, which is described by

$$S^N_{inc} = \sum_{i=1}^{n} \sum_{j=i}^{n} \frac{C_{ij}}{C_{min}} \omega_{dimer_{ij}}, \tag{2.40}$$

$$S^C_{inc} = \sum_{i=1}^{n} \sum_{j=i}^{n} C_{ij} \omega_{dimer_{ij}}, \tag{2.41}$$

$$S_{inc}^{H} = \sum_{i=1}^{n}\sum_{j=i}^{n} H_{ij}\omega_{dimer_{ij}}, \tag{2.42}$$

where C_{ij} and H_{ij} are number of carbon and hydrogen atoms in $dimer_{ij}$, respectively. n is the number of PAHs designated as soot precursors. Note that, Equation (2.40) approximates the mass of soot as mass of carbon atoms comprising the particle.

2.8 Surface growth via PAH adsorption

The adsorption of PAHs on the surface is a major mass growth pathway of soot particles. Here, a two-step process, similar to inception, is used to address the PAH adsorption. The collision of PAH molecule leads to physically bonded, $Soot - PAH^{*}$, that is followed by chemical bond formation, and completes the adsorption process. The following reactions describes the process

$$PAH + Soot \underset{k_{rv,ad}}{\overset{k_{fw,ad}}{\rightleftharpoons}} Soot{-}PAH^{*}, \tag{2.43}$$

$$Soot{-}PAH^{*} \xrightarrow{k_{rc,ad}} Soot{-}PAH. \tag{2.44}$$

The forward rate of physical adsorption, $k_{fw,ad}$, in Equation (2.45) is computed by harmonic mean of collision frequency of soot particles and PAH molecules in free molecular and continuum regime as

$$k_{fw,ad} = \frac{\beta_{fm,ad}\cdot\beta_{cont,ad}}{\beta_{fm,ad}+\beta_{cont,ad}} Av, \tag{2.45}$$

where $\beta_{fm,ad}$ is obtained [6] as

$$\beta_{fm,ad} = 2.2\sqrt{\frac{\pi k_B T}{2}\left(\frac{1}{m_{agg}}+\frac{1}{m_{PAH}}\right)}\left(d_g + d_{PAH}\right)^2, \tag{2.46}$$

where $m_{agg} = C_{tot}\cdot W_{carbon}/(N_{agg}\cdot Av)$ is the mass of soot agglomerate. $\beta_{cont,ad}$ is computed by

$$\beta_{cont,ad} = \frac{2k_B T}{3\mu}\left[\frac{C_s(d_m)}{d_g}+\frac{C_s(d_{PAH})}{d_{PAH}}\right]\left(d_g + d_{PAH}\right), \tag{2.47}$$

$$C_s(d) = 1 + \frac{2\lambda}{d}\left[1.21 + 0.4e^{(-0.78d/\lambda)}\right]\left(d_g + d_{PAH}\right). \tag{2.48}$$

The reverse rate of physical adsorption, $k_{rv,ad}$, is computed similar to reverse physical inception rate (Equation (2.35)) as

$$k_{rv,ad} = k_{fw,ad}10^{-b}e^{-a\epsilon ln(10)/(RT)}, \tag{2.49}$$

$$\epsilon = c\frac{MW_{soot}\cdot MW_{PAH}}{MW_{soot}+MW_{PAH}} - d, \tag{2.50}$$

where $MW_{soot} = C_{tot}\cdot W_{carbon}/N_{agg}$ is the equivalent molecular weight of soot, and a, b, c, and d have the same values as Equation (2.35).

The rate of chemical adsorption, $k_{rc,ad}$ is defined in the Arrhenius form [6] as

$$k_{rc,ad} = 2\times 10^{10}\cdot e^{(-96232/RT)}. \tag{2.51}$$

The total adsorption rate can be calculated assuming a steady-state concentration for physically adsorbed PAH on soot, $\partial[Soot - PAH^{*}]/\partial t = 0$, similar to inception rate (Equation (2.39)) as

$$\omega_{pah,ad} = k_{rc,ad} \frac{k_{fw,ad}[\text{Soot}][\text{PAH}]}{k_{rv,ad} + k_{rc,ad}}, \tag{2.52}$$

$$[\text{Soot}] = \rho N_{agg}. \tag{2.53}$$

The contribution of PAH adsorption rate to particle carbon and hydrogen content is computed as

$$S^C_{grow|ad} = \sum_{i=1}^{n} C_{PAH,i} \cdot \omega_{pah,ad,i}, \tag{2.54}$$

$$S^H_{grow|ad} = \sum_{i=1}^{n} (H_{PAH,i} - 2) \cdot \omega_{pah,ad,i}. \tag{2.55}$$

The rate of removal of PAH from gas mixture due to adsorption is given as

$$\omega_{PAH,i} = -\omega_{pah,ad,i}. \tag{2.56}$$

The total growth rate is sum of growth due by HACA and adsorption as

$$S^C_{grow} = S^C_{grow|HACA} + S^C_{grow|ad}, \tag{2.57}$$

$$S^H_{grow} = S^H_{grow|HACA} + S^H_{grow|ad}. \tag{2.58}$$

Chapter 3

Implementation

3.1 Overview

This chapter gives general instructions on how to implement the new library and compile it into OpenFOAM combustion models. Note that, the included code was developed at the time of writing the report, and it might change in the future. The reader is encouraged to refer to github repository [1] for updated version. The implementation steps (sections and subsections) are not designed in the sequential order, but rather in a manner that makes it easy for the reader to connect the parts of the code with different parts of the theory of soot model. Note that, the whole code is shown in the chapter. The symbol `...` in the code listings means that part of code is skipped. Reader can refer to the supplementary material that accompanies this report to access the full code.

Let us clarify some of the naming conventions specific to this library. The member data that starts with `PAH` stores information of PAH (soot precursors). `dimer` indicate the member data related to dimer (combination of two PAH molecules). `species` refers to all species used in HACA, inception and adsorption. `index` refers to the index of species in the species list in the thermodynamic file provided by user in `constant` directory. `id` denotes either the index of a species in `speciesList_` (saved in `speciesIds_`) or index of a PAH in `PAH_Names_`.

Figure 3.1 illustrates the general workflow of `laminarSoot`. Black color indicates functions and data that exist in the original library and red color denotes the parts implemented by this work. The main procedures are added to `correct()` function, and the reaction rates and heat release are passed to `reactingFoam`.

3.2 Creating the base library

The implementation begins with making a copy of `laminar` directory and renaming it to `laminarSoot` in `$FOAM_SRC/combustionModels` path, and modifying the library name using following commands

```
foam
mkdir $WM_PROJECT_USER_DIR/src/laminarSoot
cp -r $FOAM_SRC/combustionModels/laminar $WM_PROJECT_USER_DIR/src/laminarSoot
cd $WM_PROJECT_USER_DIR/src/laminarSoot
mv laminar.H laminarSoot.H
mv laminar.C laminarSoot.C
mv laminars.H laminarSoots.H
sed -i s/"laminar"/"laminarSoot"/g *.*
```

Note that, the new library should include a `Make/files` and `Make/options`

`Make/files`

```
laminarSoots.C
```

[1] https://github.com/mohammadadib-cu/laminarSoot

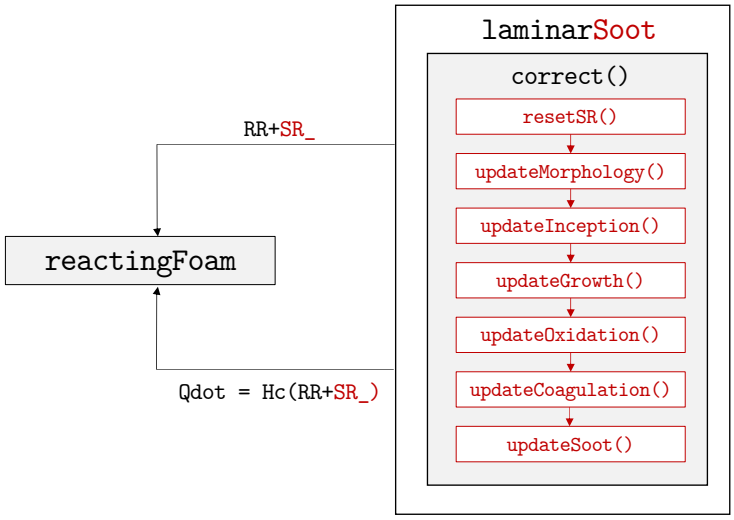

Figure 3.1: The general procedure of how `laminarSoot` works and communicates with `reactingFoam`

```

LIB = $(FOAM_USER_LIBBIN)/liblaminarSoot
```

Make/options

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/transportModels/compressible/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/basic/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/specie/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/reactionThermo/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/chemistryModel/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/turbulenceModels/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/compressible/lnInclude \
    -I$(LIB_SRC)/combustionModels/lnInclude \

LIB_LIBS = \
    -lfiniteVolume \
    -lmeshTools \
    -lcompressibleTransportModels \
    -lspecie \
    -lfluidThermophysicalModels \
    -lreactionThermophysicalModels \
    -lchemistryModel \
    -lturbulenceModels \
    -lcompressibleTurbulenceModels
```

First, two necessary header files are added to `laminarSoot.C` using the `include` statement as

laminarSoot.C

```
#include "laminarSoot.H"
#include "fvmSup.H"
#include "localEulerDdtScheme.H"
//-addtional header files
#include "IFstream.H"
#include "fvOptions.H"
```

`"IFstream.H"` is used to read thermodynamic file of reaction mechanism to extract PAH molecules information, and `"fvOptions.H"` is used in solving transport equations.

3.3 Model Constants

We declare the constants used in the soot model as static members of `laminarSoot` class in `laminarSoot.H` after the definition of the class

`laminarSoot.H`

```
template<class ReactionThermo>
class laminarSoot
:
    public ChemistryCombustion<ReactionThermo>
{
    // Static data
    static const label pi_ = Foam::constant::mathematical::pi;
    static const Foam::dimensionedScalar Av_;
    static const Foam::dimensionedScalar kB_;
    static const Foam::dimensionedScalar Ru_;
    static const Foam::dimensionedScalar rho_soot_;
    static const Foam::dimensionedScalar W_carbon_;
    static const Foam::dimensionedScalar W_hydrogen_;
    static const Foam::dimensionedScalar C_min_;
    static const Foam::dimensionedScalar H_min_;
    static const Foam::dimensionedScalar PAH_rho_const_;
```

The value and dimension of these constants are assigned in `laminarSoot.C`. For the sake of brevity only the definition of `Av_` and `kB_` is shown, which refers to Avogadro's number and Boltzmann constant, respectively. The rest of constants are defined the same way.

`laminarSoot.C`

```
// Static data
template<class ReactionThermo> const Foam::dimensionedScalar
Foam::combustionModels::laminarSoot<ReactionThermo>::Av_ =
    Foam::dimensionedScalar(
        "Av",
        Foam::dimensionSet(0,0,0,0,-1,0,0),
        scalar(6.0221409e+23)
    );

template<class ReactionThermo> const Foam::dimensionedScalar
Foam::combustionModels::laminarSoot<ReactionThermo>::kB_ =
    Foam::dimensionedScalar(
        "kB",
        Foam::dimensionSet(1,2,-2,-1,0,0,0),
        scalar(1.38064852e-23)
    );
...
```

The mathematical symbol, static member name, value and unit for each model constant are listed in Table 3.1.

3.4 Soot Properties, PAHs, Dimers, and Species

The new library reads `sootProperties` from `constant` directory of the case. `sootProps_` is the member data referring to this dictionary declared in `laminarSoot.H` along with flags for turning on/off different functionalities in the soot model.

`laminarSoot.H`

```
    // Soot properties dictionary
    const IOdictionary sootProps_;
    // Inception Flag
    bool inception_enabled_;
    // HACA Surface growth Flags
    bool HACA_growth_enabled_;
```

Table 3.1: The description of model constants

Symbol	Static member	Value	Unit
π	`pi_`	`C++ M_PI`	-
Av	`Av_`	`6.0221409e+23`	$1/\text{mol}$
k_B	`kB_`	`1.38064852e-23`	$\frac{\text{kg}-\text{m}^2}{\text{s}^2-\text{K}}$
R	`Ru_`	`8.314462618`	$\frac{\text{kg}-\text{m}^2}{\text{s}^2-\text{K}-\text{mol}}$
ρ_{soot}	`rho_soot_`	1800	$\frac{\text{kg}}{\text{m}^3}$
W_{carbon}	`W_carbon_`	`12.011e-3`	$\frac{\text{kg}}{\text{mol}}$
$\text{W}_{\text{hydrogen}}$	`W_hydrogen_`	`1.00784e-3`	$\frac{\text{kg}}{\text{mol}}$
C_{min}	`C_min_`	378	-
H_{min}	`H_min_`	20	-
-	`PAH_rho_const_`	171943.5197	mol/m^3

```
    // Surface oxidation Flag
    bool HACA_oxidation_enabled_;
    // PAH Surface growth Flags
    bool PAH_growth_enabled_ ;
    // Coagulation flag
    bool coagulation_enabled_;
    // Gas scrubbing flag
    bool scrubbing_enabled_;
```

These member data are initialized in the definition of `laminarSoot` class in `laminarSoot.C` after `integrateReactionRate_`. The first argument of `IOobject` determines the file name from `constant` directory. The soot flags are initialized by `getOrDefault` function, meaning that if the flag is not available in `sootProperties`, the default value provided as the second argument of `getOrDefault` is used (`true` in this case).

`laminarSoot.C`

```
    sootProps_
    (
        IOobject
        (
            "sootProperties",
            this->mesh().time().constant(),
            this->mesh(),
            IOobject::MUST_READ_IF_MODIFIED,
            IOobject::NO_WRITE
        )
    ),
    inception_enabled_(sootProps_.getOrDefault("inception_enabled", true)),
    HACA_growth_enabled_(sootProps_.getOrDefault("HACA_growth_enabled", true)),
    HACA_oxidation_enabled_(sootProps_.getOrDefault("HACA_oxidation_enabled", true)),
    PAH_growth_enabled_(sootProps_.getOrDefault("PAH_growth_enabled", true)),
    coagulation_enabled_(sootProps_.getOrDefault("coagulation_enabled", true)),
    scrubbing_enabled_(sootProps_.getOrDefault("scrubbing_enabled", true)),
```

The library also requires a list of precursor PAHs used in inception and PAH adsorption read from `sootProperties` under the keyword `PAHs` into `PAH_names_`. This keyword is mandatory and the names of species provided must be available in the species list in the chemistry file. `PAH_names_` is initialized in the definition of laminarSoot class in `laminarSoot.C`

`laminarSoot.C`

```
    PAH_names_
    (
        sootProps_.lookup("PAHs")
```

```
    ),
```

Three additional functions are called in the constructor after reading and checking the `integrateReactionRate_`, which is part of the original `laminar` library.

laminarSoot.C

```
    if (integrateReactionRate_)
    {
        Info<< "    using integrated reaction rate" << endl;
    }
    else
    {
        Info<< "    using instantaneous reaction rate" << endl;
    }
    //- Additional functions
    createPAHProps();
    createDimerProps();
    createSpeciesProps();
```

Note that, all of the function must be declared in the header file.

laminarSoot.H

```
        virtual bool createPAHProps();
        virtual void createDimerProps();
        virtual void createSpeciesProps();
```

`createPAHProps` creates a dictionary, `thermoDict`, from the file given provided under `foamChemistryThermoFile` keyword loops through name in `PAH_names_`, and finds the names in `thermoDict` to determine number of carbon and hydrogen and specie's index and save them into `PAH_n_C_`, `PAH_n_H_`, `PAH_indicies_`, respectively.

laminarSoot.C

```
template<class ReactionThermo>
bool Foam::combustionModels::laminarSoot<ReactionThermo>::createPAHProps()
{
    Info << "PAH names: " << PAH_names_ << endl;
    PAH_n_C_.resize(PAH_names_.size());
    PAH_n_H_.resize(PAH_names_.size());
    PAH_indicies_.resize(PAH_names_.size());

    const ReactionThermo& thermo = this->thermo();
    const dictionary thermoDict = IFstream(fileName(thermo.lookup("foamChemistryThermoFile")).expand()
    )();

    forAll(PAH_names_, i)
    {
        PAH_indicies_[i] = this->thermo().composition().species()[PAH_names_[i]];
        const dictionary* elemsDict = thermoDict.subDict(PAH_names_[i]).findDict("elements");
        wordList elemNames(elemsDict->toc());
    ...
```

`createDimerProps` creates possible dimer combinations from `PAH_names_` and saves number of carbon, and hydrogen atoms of each dimer as well as index and id of PAHs in the dimer.

laminarSoot.C

```
template<class ReactionThermo>
bool Foam::combustionModels::laminarSoot<ReactionThermo>::createDimerProps()
{
    ...
    for (int i = 0; i < PAH_names_.size(); i++) {
        for (int j = i; j < PAH_names_.size(); j++) {
            dimer_id += 1;
            dimer_names_[dimer_id] = PAH_names_[i] + PAH_names_[j];
            dimer_n_C_[dimer_id] = PAH_n_C_[i] + PAH_n_C_[j];
```

```
            dimer_n_H_[dimer_id] = PAH_n_H_[i] + PAH_n_H_[j];
            dimer_PAH_1_index_[dimer_id] = composition.species()[PAH_names_[i]];
            dimer_PAH_2_index_[dimer_id] = composition.species()[PAH_names_[j]];
            dimer_PAH_1_id_[dimer_id] = i;
            dimer_PAH_2_id_[dimer_id] = j;
        }
    }
    ...
```

`createSpeciesProps` creates a list of all species, which is a combination of `PAH_Names_` and `HACASpeciesList_` (species used in HACA mechanism). It stores the name and id of species in `speciesList_` and `speciesIds_`, respectively. After that, it initializes `SR_`, which is a list of pointers to `volScalarField`s that store the rate removal/addition of species from/to gaseous mixture. The role of `SR_` is further explained in Section 3.8.

laminarSoot.C

```
template<class ReactionThermo>
void Foam::combustionModels::laminarSoot<ReactionThermo>::createSpeciesProps()
{
    basicSpecieMixture& composition = this->thermo().composition();

    forAll(PAH_names_,i){
        speciesList_[i] = PAH_names_[i];
    }
    forAll(HACASpeciesList_,i){
        label spid = i + PAH_names_.size();
        speciesList_[spid]= HACASpeciesList_[i];
    }
    Info << "List of species: " << speciesList_ << endl;


    forAll(speciesList_, i)
    {
        const label specieIndex = composition.species()[speciesList_[i]];
        if (!composition.species().found(speciesList_[i]))
        {
            FatalIOErrorIn("laminarSoot::findIndicies()", this->thermo())
                << speciesList_[i] <<" is not found in available species "
                << composition.species() << exit(FatalIOError);
        }
        speciesIndicies_.insert
        (
            speciesList_[i],
            specieIndex
        );
        speciesIds_.insert
        (
            speciesList_[i],
            i
        );
        Info << speciesList_[i] << " is found! Index= " << speciesIndicies_[speciesList_[i]] << " Id:
     " << speciesIds_(speciesList_[i]) << endl;
    }

    // Create the fields for the chemistry sources
    forAll(SR_, fieldi)
    {
        SR_.set
        (
            fieldi,
            new volScalarField::Internal
            (
                IOobject
                (
                    "SR." + speciesList_[fieldi],
                    this->mesh().time().timeName(),
```

```
50                      this->mesh(),
51                      IOobject::NO_READ,
52                      IOobject::NO_WRITE
53                  ),
54                  this->mesh(),
55                  dimensionedScalar(dimMass/dimVolume/dimTime, Zero)
56              )
57          );
58      }
59  }
```

3.5 Tracked and Derived Fields

Here, we declare and initialize the four *tracked* fields, N_{agg}, N_{pri}, C_{tot}, H_{tot}, and fields *derived* from tracked fields and thermodynamic properties of gas mixture. These fields describe soot morphology (needs to be written at each time step) and source terms. The declaration are added to `laminarSoot.H` after the scrubbing flag.

`laminarSoot.H`

```
1  ...
2      // Soot tracked fields
3      volScalarField N_agg_;
4      volScalarField N_pri_;
5      volScalarField C_tot_;
6      volScalarField H_tot_;
7      // Morphology
8      volScalarField n_p_;
9      volScalarField d_p_;
10     volScalarField d_m_;
11     volScalarField d_g_;
12     volScalarField A_tot_;
13     // Source Terms
14     // Inception
15     volScalarField S_inc_N_;
16     volScalarField S_inc_C_tot_;
17     volScalarField S_inc_H_tot_;
18     // Surface growth
19     volScalarField S_grow_C_tot_;
20     volScalarField S_grow_H_tot_;
21     // Oxidation
22     volScalarField S_ox_C_tot_;
23     // Coagulation
24     volScalarField S_coag_N_agg_;
25 ...
```

The main tracked fields are read from the first time directory (`0`), so they are initialized using `MUST_READ` keyword. Here, only the definition of `N_agg_` (corresponds to N_{agg} in the model) is provided in the code below, and the rest of tracked fields are defined similarly.

`laminarSoot.C`

```
1  ...
2      N_agg_
3      (
4          IOobject
5          (
6              "N_agg",
7              this->mesh().time().timeName(),
8              this->mesh(),
9              IOobject::MUST_READ,
10             IOobject::AUTO_WRITE
11         ),
12         this->mesh()
13     ),
```

```
...
```

However, the derived fields and source terms are only written in the time directories by running the case, so they are initialized using `NO_READ` keyword. Here, the initialization of `n_p_` (corresponds to n_p in the model) is given.

laminarSoot.C

```
...
    n_p_
    (
        IOobject
        (
            "n_p",
            this->mesh().time().timeName(),
            this->mesh(),
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        this->mesh(),dimensionedScalar("n_p", dimensionSet(0,0,0,0,0,0,0),1.0)
    ),
...
```

3.5.1 Tracked field functions

Four additional functions are defined in `laminarSoot.H` with similar names to tracked fields to return the tracked fields.

laminarSoot.H

```
        //- Return the number concentration of agglomerates N_agg
        // Units: mol/kg
        virtual tmp<volScalarField> N_agg() const
        {
            return N_agg_;
        };
        //- Return the number concentration of primary particles N_pri
        // Units: mol/kg
        virtual tmp<volScalarField> N_pri() const
        {
            return N_pri_;
        };
        //- Return the carbon contant of soot C_tot
        // Units: mol/kg
        virtual tmp<volScalarField> C_tot() const
        {
            return C_tot_;
        };
        //- Return the carbon contant of soot H_tot
        // Units: mol/kg
        virtual tmp<volScalarField> H_tot() const
        {
            return H_tot_;
        };
```

3.6 Gas properties

Some of member functions of `laminarSoot` are used to calculate additional gas properties used by the soot model. These functions are declared and defined in `laminarSoot.H`

3.6.1 W(index) function

This function receives the index of a specie and returns the molecular weight in kg/mol as a `dimensionedScalar`. Note that, `composition.W(index)` returns a scalar (non-dimensional) equal to molecular weight of a specie in `kmol/kg`.

laminarSoot.H

```
        virtual dimensioned<scalar> W(label index)
        {
            basicSpecieMixture& composition = this->thermo().composition();
            return composition.W(index) / 1000.0 * dimensionedScalar(dimensionSet(1,0,0,0,-1,0,0),
    scalar(1));
        };
}
```

3.6.2 C(index) function

This function takes the index of a specie as an argument and returns the concentration as a `tmp<volScalarField>` from mass fraction, `Y`, using the ideal gas law.

laminarSoot.H

```
        virtual tmp<volScalarField> C(label index)
        {
            basicSpecieMixture& composition = this->thermo().composition();
            return tmp<volScalarField>
            (
                new volScalarField
                (
                    "C",
                    (composition.Y()[index] * this->thermo().rho() / W(index))
                )
            );
        };
}
```

3.6.3 lambda_gas() function

This function calculates mean free path of gas (corresponding to Equation (2.12)).

laminarSoot.H

```
        virtual tmp<volScalarField> lambda_gas() const
        {
            return tmp<volScalarField>
            (
                new volScalarField
                (
                    max
                    (
                        this->thermo().mu() / this->thermo().rho() * pow(pi_ * this->thermo().W() /
    (2.0* kB_ * Av_ * this->thermo().T()), 0.5),
                        dimensionedScalar(dimensionSet(0,1,0,0,0,0,0), SMALL)
                    )
                )
            );
        };
}
```

3.7 PAH Characteristics

3.7.1 m_PAH(id) function

This function takes the id of a PAH and returns the mass of one PAH molecule according to Equation (2.31).

laminarSoot.H

```
Info<< "Reading field N_agg\n" << endl;
        virtual dimensioned<scalar> m_PAH(label id)
        {
            label index = PAH_indicies_[id];
            return W(index) / Av_;
        };
}
```

3.7.2 d_PAH(id) function

This function takes the id of a PAH and returns diameter of one PAH molecule according to Equations 2.32-2.34.

laminarSoot.H

```
        virtual dimensioned<scalar> d_PAH(label id)
        {
            const dimensionedScalar rho_PAH = PAH_rho_const_ * (W_carbon_ * PAH_n_C_[id] + W_hydrogen_
     * PAH_n_H_[id]) / (PAH_n_C_[id] +  PAH_n_H_[id]);
            const dimensionedScalar V_PAH = m_PAH(id) / rho_PAH;
            return pow(6.0 * V_PAH / pi_, 1.0/3.0);
        };
```

3.8 Modifying R(Y) function

R(Y) takes the mass traction field of a specie, Y, as an argument and returns its production/consumption rate in kg.m^{-3}.s. We modified this function in such a way that it looks for the name of provided species (accessible through Y.member()) in speciesList_, and if available, SR_ of that specie is added to Su. Note that, RR() return the rate of production/destruction of species calculated using the reactions defined in the chemistry file (reaction mechanism) in the Arrhenius form. However, soot inception, growth (via HACA and adsorption) and oxidation also transforms a fraction of species into soot mass or releases some species into gas mixture. In essence, SR_ accounts for the rate of addition/removal of any specie involved in the soot formation.

laminarSoot.C

```
template<class ReactionThermo>
Foam::tmp<Foam::fvScalarMatrix>
Foam::combustionModels::laminarSoot<ReactionThermo>::R(volScalarField& Y) const
{
    tmp<fvScalarMatrix> tSu(new fvScalarMatrix(Y, dimMass/dimTime));

    fvScalarMatrix& Su = tSu.ref();

    if (this->active())
    {
        const label specieI =
            this->thermo().composition().species()[Y.member()];

        Su += this->chemistryPtr_->RR(specieI);
        // scrubbing rates are added
        if (scrubbing_enabled_){
            if (speciesList_.found(Y.member()))
```

```
            {
                label spid = speciesIds_[Y.member()];
                Su += SR_[spid];
            }
        }
    }

    return tSu;
}
```

3.9 Modifying Qdot() function

Qdot calculates the rate of energy change by production/consumption of species due to chemical reactions. Here, we modify this function to incorporate the contribution of addition/removal of species by soot formation to energy release. The process is similar to modification of R(Y) function. The rate of energy change is calculated by multiplying chemical enthalpy (this->thermo().composition().Hc(i)) to sum of RR and SR_.

laminarSoot.C

```
template<class ReactionThermo>
Foam::tmp<Foam::volScalarField>
Foam::combustionModels::laminarSoot<ReactionThermo>::Qdot() const
{
    tmp<volScalarField> tQdot
    (
        new volScalarField
        (
            IOobject
            (
                this->thermo().phasePropertyName(typeName + ":Qdot"),
                this->mesh().time().timeName(),
                this->mesh(),
                IOobject::NO_READ,
                IOobject::NO_WRITE,
                false
            ),
            this->mesh(),
            dimensionedScalar(dimEnergy/dimVolume/dimTime, Zero)
        )
    );

    if (this->active())
    {
        if (scrubbing_enabled_){
            scalarField& Qdot = tQdot.ref();
            forAll(this->thermo().composition().Y(), i)
            {
                if (speciesList_.found(this->thermo().composition().Y()[i].member()))
                {
                    label spid = speciesIds_[this->thermo().composition().Y()[i].member()];
                    forAll(Qdot, celli)
                    {
                        const scalar hi = this->thermo().composition().Hc(i);
                        Qdot[celli] -= hi*(this->chemistryPtr_->RR(i)[celli]+SR_[spid][celli]);
                    }
                }else{
                    forAll(Qdot, celli)
                    {
                        const scalar hi = this->thermo().composition().Hc(i);
                        Qdot[celli] -= hi*this->chemistryPtr_->RR(i)[celli];
                    }
                }
            }
        }else{
```

```
            tQdot.ref() = this->chemistryPtr_->Qdot();
        }
    }
    return tQdot;
}
```

3.10 Modifying correct() function

correct function is called each time before solving the species concentration, YEqn, by reactingFoam. We add functions that needs to be called at each iteration to calculate the source terms and solves the soot equations. These functions must be called in the correct order to produced the precises results.

laminarSoot.C

```
Info<< "Reading field N_agg\n" << endl;
template<class ReactionThermo>
void Foam::combustionModels::laminarSoot<ReactionThermo>::correct()
{
...
        //- additional functions
        resetSR();
        updateMorphology();
        updateInception();
        updateGrowth();
        updateOxidation();
        updateCoagulation();
        updateSoot();
...
}
```

Note that, all of these functions must be declared in the header file.

laminarSoot.H

```
        virtual void resetSR();
        virtual void updateMorphology();
        virtual void updateInception();
        virtual void updateGrowth();
        virtual void updateOxidation();
        virtual void updateCoagulation();
        virtual void updateSoot();
```

3.11 Resetting scrubbing rates by resetSR()

resetSR loops through fields inserted in SR_, and then iterates through cells of each fields and zeros the values of that field.

laminarSoot.C

```
template<class ReactionThermo>
void Foam::combustionModels::laminarSoot<ReactionThermo>::resetSR()
{
    if (scrubbing_enabled_){
        forAll(SR_, fieldi){
            forAll(SR_[fieldi], celli){
                SR_[fieldi][celli] = 0.0;
            }
        }
    }
}
```

3.12 Updating soot morphology

3.12.1 V_agg() function

This function returns the volume of agglomerates.

laminarSoot.C

```
        virtual tmp<volScalarField> V_agg()
        {
            return tmp<volScalarField>
            (
                new volScalarField
                (
                    C_tot() * W_carbon_ / (rho_soot_ * N_agg() * Av_)
                )
            );
        };
```

3.12.2 m_agg() function

This function returns the mass of agglomerates.

laminarSoot.C

```
        virtual tmp<volScalarField> m_agg()
        {
            return tmp<volScalarField>
            (
                new volScalarField
                (
                    C_tot() * W_carbon_ / (N_agg() * Av_)
                )
            );
        };
```

3.12.3 updateMorphology() function

`updateMorphology` uses the tracked fields to calculate derived fields that characterize the soot morphology n_p, d_p, d_m, and d_m. Note that, `n_p.max(1.0)` enforces $n_p \geq 1$ condition. This function is related to Equations 2.5-2.8 in Section 2.2.

laminarSoot.C

```
template<class ReactionThermo>
void Foam::combustionModels::laminarSoot<ReactionThermo>::updateMorphology()
{
    // number of primary particles
    n_p_ = N_pri() / N_agg();
    n_p_.max(1.0);
    if (!coagulation_enabled_)
    {
        Info<< "enforcing n_p = 1\n" << endl;
        n_p_.min(1.00000001);
    }
    // Primary particle diameter
    d_p_ = pow (
        (6.0 / pi_) *
        (C_tot() * W_carbon_) / rho_soot_ *
        1.0 / (N_pri() * Av_)
        , 1.0/3.0
    );
    // Surface area of each primary particle
    A_tot_ = N_pri() * Av_ * pi_ * d_p_ * d_p_;
    //  Mobility diameter
```

```
    d_m_ = d_p_ * pow(n_p_, 0.45);
    // Gyration Diameter
    volScalarField n_p_lowerlimit (n_p_*0.0+1.5);
    d_g_ = (n_p_ <= n_p_lowerlimit) * (d_m_ / 1.29) + (n_p_ > n_p_lowerlimit) * (d_m_ / (pow(n_p_,
     -0.2)+0.4));

}
```

3.13 Updating Inception Source Term

3.13.1 `updateInception()` fuction

`updateInception()` function loops through the dimers and passes the id of PAHs in the dimer as arguments to `dimerROP` function to calculate the rate of formation of chemically-bonded dimer. Then, the contribution of each dimer to source terms, `S_inc_N_` , `S_inc_C_tot_` , and `S_inc_H_tot_` is computed according to Equations 2.40-2.42. After that, the rate scrubbing rate of PAH and hydrogen (released by carbonization of dimer) is added to relevant fields in `SR_`.

laminarSoot.C

```
template<class ReactionThermo>
void Foam::combustionModels::laminarSoot<ReactionThermo>::updateInception()
{
    S_inc_N_ *= 0.0;
    S_inc_C_tot_ *= 0.0;
    S_inc_H_tot_ *= 0.0;
    if (inception_enabled_){
        volScalarField rho = this->thermo().rho();
        forAll(dimer_names_, i)
        {
            // PAH Index and Id
            label id1 = dimer_PAH_1_id_[i];
            label id2 = dimer_PAH_2_id_[i];
            volScalarField dimerROPField(dimerROP(id1, id2));
            // N_agg Source Term
            S_inc_N_ += (dimer_n_C_[i] / C_min_) * dimerROPField / rho;
            // C_tot Source Term
            S_inc_C_tot_ += dimer_n_C_[i] * dimerROPField / rho;
            // H_tot Source Term
            S_inc_H_tot_ += dimer_n_H_[i] * dimerROPField / rho;
            if (scrubbing_enabled_){
                // PAHs
                // species id
                label spid1 = speciesIds_[PAH_names_[id1]];
                label spid2 = speciesIds_[PAH_names_[id2]];
                // species index
                label spindex1 = speciesIndicies_[PAH_names_[id1]];
                label spindex2 = speciesIndicies_[PAH_names_[id2]];
                SR_[spid1] -= dimerROPField * W(spindex1);
                SR_[spid2] -= dimerROPField * W(spindex2);

                // H2
                label H2_id = speciesIds_["H2"];
                label H2_index = speciesIndicies_["H2"];
                SR_[H2_id] += dimerROPField * W(H2_index);
            }
        }
    }
}
```

3.13.2 `dimerROP(id1, id2)` function

This function calculates the rate of formation of dimer chemically bonded dimer according to Equation (2.39). The rate of dimer formation cannot be negative, so `max` function is used to enforce this condition.

lamiarSoot.H

```
        virtual tmp<volScalarField> dimerROP(label id1, label id2)
        {
            label index1 = PAH_indicies_[id1];
            label index2 = PAH_indicies_[id2];
            return tmp<volScalarField>
            (
                new volScalarField
                (
                    "dimerROP",
                    max
                    (
                        k_REAC() * k_FWD(id1, id2) * C(index1) * C(index2) / (k_REAC() + k_REV(id1,
    id2)),
                        dimensionedScalar(dimensionSet(0,-3,-1,0,1,0,0), scalar(0.0))
                    )
                )
            );
        }
```

3.13.3 `k_FWD(id1, id2)` funtion

This function calculates the forward rate constant for physical dimerization of two PAHs according to Equation (2.30).

lamiarSoot.H

```
        virtual tmp<volScalarField> k_FWD(label id1, label id2)
        {
            const volScalarField& T = this->thermo().T();
            // reduced dimer diameter
            const dimensionedScalar d_reduced = 2 * d_PAH(id1) * d_PAH(id2) / (d_PAH(id1) + d_PAH(id2)
    );
            // reduced dimer mass
            const dimensionedScalar m_reduced = m_PAH(id1) * m_PAH(id2) / (m_PAH(id1) + m_PAH(id2));
            return tmp<volScalarField>
            (
                new volScalarField
                (
                    "k_FWD",
                    (1.0 * 2.2 * 0.1 * Av_ * d_reduced * d_reduced * pow(8.0 * pi_ * kB_ * T /
    m_reduced, 0.5))
                )
            );
        };
```

3.13.4 `k_REV(id1, id2)` funtion

This function calculates the reverse rate constant for physical dimerization of two PAHs according to Equations 2.35 and 2.36.

lamiarSoot.H

```
        virtual tmp<volScalarField> k_REV(label id1, label id2)
        {
            // Constants
            dimensionedScalar a_k_REV("a_k_REV", dimensionSet(0,0,0,0,0,0,0),  scalar(0.115));
            dimensionedScalar b_k_REV("b_k_REV", dimensionSet(0,0,0,0,0,0,0),  scalar(1.8));
```

```
            dimensionedScalar c_k_REV("c_k_REV", dimensionSet(0,2,-2,0,0,0,0), scalar(933420.0));
            dimensionedScalar d_k_REV("d_k_REV", dimensionSet(1,2,-2,0,-1,0,0),scalar(34069.0));

            const volScalarField& T = this->thermo().T();
            label index1 = PAH_indicies_[id1];
            label index2 = PAH_indicies_[id2];
            const dimensionedScalar W_reduced = W(index1) * W(index2) / (W(index1) + W(index2));
            const dimensionedScalar epsilon = c_k_REV * W_reduced - d_k_REV;

            return tmp<volScalarField>
            (
                new volScalarField
                (
                    "k_REV",
                    (k_FWD(id1, id2) * pow(10.0, -b_k_REV) * exp(-a_k_REV * epsilon * 2.3025851/(Ru_ *
    T)))
                )
            );
        };
```

3.13.5 k_REAC() function

This function calculates the rate constant for chemical dimerization of two PAHs according to Equation (2.37).

lamiarSoot.H

```
        virtual tmp<volScalarField> k_REAC()
        {
            // Chemical dimerization rate constants
            dimensionedScalar Ea_k_REAC("Ea_k_REAC", dimensionSet(1,2,-2,0,-1,0,0), scalar(96232.0));
            dimensionedScalar A_k_REAC("A_k_REAC", dimensionSet(0,3,-1,0,-1,0,0), scalar(5.0e6));
            const volScalarField& T = this->thermo().T();

            return tmp<volScalarField>
            (
                new volScalarField
                (
                    "k_REAC",
                    (A_k_REAC * exp(-Ea_k_REAC/(Ru_ * T)))
                )
            );
        }
```

3.14 Updating soot growth rates

3.14.1 updateGrowth() function

`updateGrowth()` calculates the growth source terms for transport equations of C_{tot} and H_{tot} in two steps. First, it calculates the HACA rate by `HACAGrowthRate()`, and adds the contribution of HACA to growth source terms to Equations 2.19 and 2.20. Then, it loops over `PAH_names_`, calculates adsorption rate by `PAHAdsorptionRate()` for each PAH, and incorporates it into growth source terms. After each step, `SR_` is updated for relevant fields to account for addition/removal of species to/from gas mixtures.

lamiarSoot.C

```
template<class ReactionThermo>
void Foam::combustionModels::laminarSoot<ReactionThermo>::updateGrowth()
{
    S_grow_C_tot_ *= 0.0;
    S_grow_H_tot_ *= 0.0;

```

```
    if (HACA_growth_enabled_)
    {
        volScalarField rho = this->thermo().rho();
        volScalarField HACAGrowthRateField(HACAGrowthRate());
        S_grow_C_tot_ += 2 * HACAGrowthRateField / rho;
        S_grow_H_tot_ += 2 * HACAGrowthRateField * (0.25 / 2.00) / rho;

        if (scrubbing_enabled_){
            // C2H2
            label C2H2_id = speciesIds_["C2H2"];
            label C2H2_index = speciesIndicies_["C2H2"];
            SR_[C2H2_id] -= HACAGrowthRateField * W(C2H2_index);

            // H
            label H_id = speciesIds_["H"];
            label H_index = speciesIndicies_["H"];
            SR_[H_id] += HACAGrowthRateField * W(H_index) * (1.75 / 2.00);
        }
    }
    if (PAH_growth_enabled_)
    {
        volScalarField rho = this->thermo().rho();
        forAll(PAH_names_, id)
        {
            volScalarField PAHAdsorptionRateField(PAHAdsorptionRate(id));
            S_grow_C_tot_ += PAH_n_C_[id] * PAHAdsorptionRateField / rho;
            S_grow_H_tot_ += (PAH_n_H_[id] - 2) * PAHAdsorptionRateField / rho;

            if (scrubbing_enabled_){
                // PAH
                // species id
                label spid = speciesIds_[PAH_names_[id]];
                // species index
                label spindex = speciesIndicies_[PAH_names_[id]];
                SR_[spid] -= PAHAdsorptionRateField * W(spindex);

                // H
                label H_id = speciesIds_["H"];
                label H_index = speciesIndicies_["H"];
                SR_[H_id] += PAHAdsorptionRateField * W(H_index) * 2;
            }
        }
    }

}
```

3.14.2 `HACAGrowthRate()` function

This function computes the rate of addition of C_2H_2 molecules to soot particle surface via HACA according to Equation (2.17).

lamiarSoot.H

```
        virtual tmp<volScalarField> HACAGrowthRate()
        {
            label C2H2_i = speciesIndicies_["C2H2"];
            return tmp<volScalarField>
            (
                new volScalarField
                (
                    max
                    (
                        alpha() * k_4_HACA() * C(C2H2_i) * C_soot_0(),
                        dimensionedScalar(dimensionSet(0,-3,-1,0,1,0,0), scalar(0.0))
                    )
                )
```

```
            );
        }
```

3.14.3 Other HACA-growth-related functions

The rest of functions related to growth rate via HACA defined in `laminarSoot.H` are given in Table 3.2 with parameter name and a references to equation(s) from the soot model used in each function. The source code for these functions are provided in Appendix A.1.

Table 3.2: The description of other HACA-growth-related functions

Function	Parameter	Reference
`k_4_HACA()`	k_{f4}	Table 2.1 No.4-f
`k_5_HACA()`	k_{f5}	Table 2.1 No.5-f
`k_6_HACA()`	k_{f6}	Table 2.1 No.1-f
`alpha()`	α	Equation (2.21)
`C_soot_0()`	$[C_{soot^\circ}]$	Equation (2.22) Table 2.1 No.1-f Table 2.1 No.1-r Table 2.1 No.2-f Table 2.1 No.1-r Table 2.1 No.3-f

3.14.4 `PAHAdsorptionRate(id)` function

This function takes the PAH id and calculates the adsorption rate for the given PAH. The forward ($k_{fw,ad}$) and reverse ($k_{rv,ad}$) rates for PAH and soot collision, and chemical bond formation ($k_{rc,ad}$) between soot and PAH molecules are computed inside the function.

`lamiarSoot.H`

```
        virtual tmp<volScalarField> PAHAdsorptionRate(label id)
        {

            // PAH Index and Id
            label index = PAH_indicies_[id];
            // Temperature
            const volScalarField& T = this->thermo().T();
            // Density
            const volScalarField rho = this->thermo().rho();
            // Viscosity of gas
            volScalarField mu = this->thermo().mu();
            // beta fm coag PAH-soot
            volScalarField beta_fm_soot_PAH
            (
                2.2 * pow(pi_ * kB_ * T / 2.0 * (1.0/m_agg() + 1.0/m_PAH(id)), 0.5) * pow(d_g_ + d_PAH
    (id), 2.0)
            );
            // beta cont coag PAH-soot
            volScalarField C_s_soot_d_m
            (
                1.0 + (2.0 * lambda_gas() / d_m_ ) * (1.21 + 0.4*exp(-0.78 * d_m_ / lambda_gas()))
            );
            volScalarField C_s_soot_d_PAH
            (
                1.0 + (2.0 * lambda_gas() / d_PAH(id)) * (1.21 + 0.4*exp(-0.78 * d_PAH(id) /
    lambda_gas()))
            );
            volScalarField beta_cont_soot_PAH
            (
```

```
                2.0 * kB_ * T / (3.0 * mu) * (C_s_soot_d_m / d_g_ + C_s_soot_d_PAH / d_PAH(id)) * (
    d_g_ + d_PAH(id))
            );
            // Forward reaction rate
            volScalarField k_FWD_soot_PAH
            (
                beta_fm_soot_PAH * beta_cont_soot_PAH / (beta_fm_soot_PAH + beta_cont_soot_PAH) * Av_
            );
            // W reduced
            // Constants
            dimensionedScalar a_k_REV_soot_PAH(dimensionSet(0,0,0,0,0,0,0),  scalar(0.115));
            dimensionedScalar b_k_REV_soot_PAH(dimensionSet(0,0,0,0,0,0,0),  scalar(1.8));
            dimensionedScalar c_k_REV_soot_PAH(dimensionSet(0,2,-2,0,0,0,0), scalar(933420.0));
            dimensionedScalar d_k_REV_soot_PAH(dimensionSet(1,2,-2,0,-1,0,0),scalar(34053.0));
            // W soot
            volScalarField W_soot(C_tot() * W_carbon_ / N_agg());
            volScalarField epsilon_soot_PAH
            (
                c_k_REV_soot_PAH * (W(index) * W_soot) / (W(index) + W_soot) - d_k_REV_soot_PAH
            );
            // The reverse rate of physical dimerization
            volScalarField k_REV_soot_PAH
            (
                k_FWD_soot_PAH * pow(10.0, -b_k_REV_soot_PAH) * exp(-a_k_REV_soot_PAH *
    epsilon_soot_PAH * 2.3025851/(Ru_ * T))
            );

            // Chemical adsorption rate
            dimensionedScalar Ea_k_REAC_soot_PAH(dimensionSet(1,2,-2,0,-1,0,0), scalar(96232.0));
            dimensionedScalar A_k_REAC_soot_PAH(dimensionSet(0,3,-1,0,-1,0,0), scalar(2.0e10));
            volScalarField k_REAC_soot_PAH
            (
                A_k_REAC_soot_PAH * exp(-Ea_k_REAC_soot_PAH / (Ru_ * T))
            );
            return tmp<volScalarField>
            (
                new volScalarField
                (
                    max
                    (
                        k_REAC_soot_PAH * k_FWD_soot_PAH * C(index) * (N_agg() * rho) / (
    k_REAC_soot_PAH + k_REV_soot_PAH),
                        dimensionedScalar(dimensionSet(0,-3,-1,0,1,0,0), scalar(0.0))
                    )
                )
            );
        }
```

3.15 Updating oxidation rates

3.15.1 `updateOxidation()` function

This function calculates the rate of oxidation of soot particles by O_2 (`HACAO2OxidationRate`) and OH (`HACAOHOxidationRate`) and adds their contribution to oxidation source term. Also, the rate of removal of O_2 and OH and addition of CO to the gas mixture are added to `SR_`.

lamiarSoot.C

```
template<class ReactionThermo>
void Foam::combustionModels::laminarSoot<ReactionThermo>::updateOxidation()
{
    S_ox_C_tot_ *= 0.0;
    if (HACA_oxidation_enabled_){
        volScalarField rho(this->thermo().rho());
        volScalarField HACAO2OxidationRateField(HACAO2OxidationRate());
```

```
        volScalarField HACAOHOxidationRateField(HACAOHOxidationRate());
        S_ox_C_tot_ += -1 * (HACAO2OxidationRateField + HACAOHOxidationRateField) / rho;

        if (scrubbing_enabled_){
            // O2
            label O2_id = speciesIds_["O2"];
            label O2_index = speciesIndicies_["O2"];
            SR_[O2_id] -= 0.5 * HACAO2OxidationRateField * W(O2_index);

            // CO2
            label CO_id = speciesIds_["CO"];
            label CO_index = speciesIndicies_["CO"];
            SR_[CO_id] += (HACAO2OxidationRateField + HACAOHOxidationRateField) * W(CO_index);

            // OH
            label OH_id = speciesIds_["OH"];
            label OH_index = speciesIndicies_["OH"];
            SR_[OH_id] -= HACAOHOxidationRateField * W(OH_index);
        }
    }
}
```

3.15.2 `HACAO2OxidationRate()` function

This function returns the rate of oxidation by O_2 according to Equation (2.25).

lamiarSoot.H

```
        virtual tmp<volScalarField> HACAO2OxidationRate()
        {
            label O2_i = speciesIndicies_["O2"];
            volScalarField rho (this->thermo().rho());
            return tmp<volScalarField>
            (
                new volScalarField
                (
                    max
                    (
                        2 * alpha() * k_5_HACA() * C(O2_i) * C_soot_0(),
                        dimensionedScalar(dimensionSet(0,-3,-1,0,1,0,0), scalar(0.0))
                    )
                )
            );
        }
```

3.15.3 `HACAOHOxidationRate()` function

This function returns the rate of oxidation by OH according to Equation (2.26).

lamiarSoot.H

```
        virtual tmp<volScalarField> HACAO2OxidationRate()
        {
            label O2_i = speciesIndicies_["O2"];
            volScalarField rho (this->thermo().rho());
            return tmp<volScalarField>
            (
                new volScalarField
                (
                    max
                    (
                        2 * alpha() * k_5_HACA() * C(O2_i) * C_soot_0(),
                        dimensionedScalar(dimensionSet(0,-3,-1,0,1,0,0), scalar(0.0))
                    )
                )
```

```
            );
        }
```

3.16 Updating coagulation rate by updateCoagulation()

updateCoagulation() determines the coagulation source term S^N_{coag}. First, it calculates the collision frequency in free molecular (β_{fm}) and continuum (β_{cont}) regime using Equations 2.15 and 2.16, respectively. Then, it uses harmonic mean of collision frequencies considering %82 enhancement factor (Equation (2.14)) and returns the rate coagulation (Equation (2.13)).

lamiarSoot.C

```
template<class ReactionThermo>
void Foam::combustionModels::laminarSoot<ReactionThermo>::updateCoagulation()
{
    const volScalarField& T = this->thermo().T();
    volScalarField mu (this->thermo().mu());
    volScalarField rho (this->thermo().rho());
    // Free Molecule
    volScalarField beta_fm
    (
        4 * pow(pi_ * kB_ * T / m_agg(), 0.5) * d_g_ * d_g_
    );
    // Continuum
    volScalarField beta_cont(
        (8 * kB_ / (3 * mu)) * T * ( 1.0 + (2.0 * lambda_gas() / d_m_ )*(1.21 + 0.4*exp(-0.78*d_m_/
     lambda_gas())))
    );
    // Coagulation source term
    if (coagulation_enabled_){
        S_coag_N_agg_ = 0.5 * 1.82 * beta_fm * beta_cont / (beta_fm + beta_cont) * pow(N_agg(), 2.0) *
      Av_ * rho;
    }
}
```

3.17 Solving the transport equations

3.17.1 diffusionCoeff function

This function returns the diffusion coefficient of soot particles used in transport equations of tracked fields. The calculations are based on Eqautions 2.9-2.11.

lamiarSoot.H

```
        virtual tmp<volScalarField> diffusionCoeff()
        {
            volScalarField mu (this->thermo().mu());
            const volScalarField& T = this->thermo().T();
            volScalarField f_gas_particle
            (
                3.0 * pi_ * mu * d_m_ /
                ( 1.0 + 2.0 * lambda_gas() / d_m_ *(1.21 + 0.4*exp(-0.78*d_m_/lambda_gas())))
            );

            f_gas_particle.max(SMALL);

            return tmp<volScalarField>
            (
                new volScalarField
                (
                    kB_ * T / f_gas_particle
                )
```

```
            );
        }
```

3.17.2 `updateSoot()` function

This function deals with the transport equation of tracked fields by defining `tmp<fvScalarMatrix>` for each field and solving them. Below only `N_aggEqn` is shown, and the rest equations are treated in a similar way. Note that, `S_coag_N_agg_` is always negative, so this source term is added implicitly using `fvm::Sp` function to the equation matrix, but `S_inc_N_` is positive and added explicitly.

lamiarSoot.C

```
template<class ReactionThermo>
void Foam::combustionModels::laminarSoot<ReactionThermo>::updateSoot()
{
    const surfaceScalarField& phi = this->phi();
    const volScalarField D(diffusionCoeff());
    volScalarField rho (this->thermo().rho());
    fv::options& fvOptions(fv::options::New(this->mesh_));

    // N_agg Equation
    {
        Info<< "N_agg Equation \n" << endl;
        tmp<fvScalarMatrix> N_aggEqn
        (
            fvm::ddt(rho, N_agg_)
            + fvm::div(phi, N_agg_)
            - fvm::laplacian(D*rho, N_agg_)
        ==
            - fvm::Sp(rho * S_coag_N_agg_ /N_agg_, N_agg_)
            + rho * S_inc_N_
        );

        N_aggEqn.ref().relax();
        fvOptions.constrain(N_aggEqn.ref());
        solve(N_aggEqn);
        fvOptions.correct(N_agg_);
    }
...
```

3.18 Compiling the library

`wmake` complies the library by building `laminarSoot.o` and linking it to the share combustion models library.

terminal output

```
wmake
...
laminarSoot/laminarSoots.C -o /home/mo/OpenFOAM/OpenFOAM-v2006/build/linux64GccDPInt32Opt/src/
    combustionModels/laminarSoot/laminarSoots.o
...
-L/home/mo/OpenFOAM/OpenFOAM-v2006/platforms/linux64GccDPInt32Opt/lib \
    -lfiniteVolume -lmeshTools -lcompressibleTransportModels -lspecie -lfluidThermophysicalModels -
    lreactionThermophysicalModels -lchemistryModel -lturbulenceModels -lcompressibleTurbulenceModels
     -o /home/mo/OpenFOAM/OpenFOAM-v2006/platforms/linux64GccDPInt32Opt/lib/libcombustionModels.so
```

Chapter 4

Tutorial set up

4.1 Physics and Geometry

A 0D constant volume reactor case is designed to test `laminarSoot` combustion model. The geometry is a 2D 0.1m×0.1m closed box. The walls are assumed to be insulated ($\partial T/\partial x = 0$). The gas mixture inside the box is initially T=1800 K and P=10^5 kPa with the composition of $Y_{C_2H_4} = 0.2$, $Y_{N_2} = 0.8$. At this temperature, C_2H_4 decomposes through a set of chemical reactions and forms PAHs that generate soot particles inside the box. The residence time of gas is 400 ms (0.4 s). In other words, the simulation continues for 400 ms. There are no spatial gradients throughout the domain. Figure 4.1 shows schematics of the reactor with boundary names. The simple physics and small number of grid points offer advantages in term of (i) computational cost that could be enormous for reacting mixture considering 101 species transport equations (in case of ABF mechanism), and (ii) validation that will be done against the results of *EPTLSOOT* code developed by the author for modelling soot using low dimensional reactors based on Cantera [13] kinetic solver and Scipy [21] ODE integration libraries.

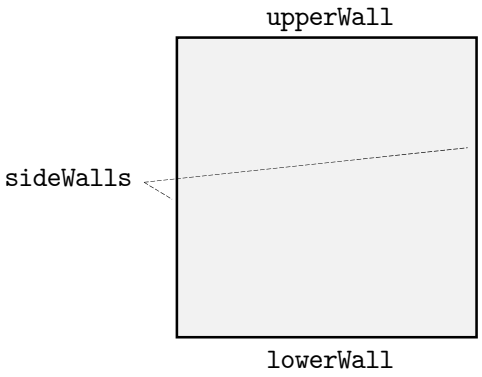

Figure 4.1: The schematics of reactor with boundary names

The mesh file, `reactor.msh`, is created by *Gambit*, and should be placed inside the case directory.

4.2 Combustion Model, Chemistry, and Theromphysical Properties

To utilize the new combustion model, user needs to set `combustionModel` to `laminarSoot` in `constant/combustionProperties`.

`constant/combustionProperties`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2006                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      combustionProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

combustionModel  laminarSoot;

active  true;

laminarCoeffs
{
}

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

`sootProperties` also must be added to `constant` directory. The name of soot precursors, `PAHs`, is the only mandatory keyword, which is provided in the file.

`constant/sootProperties`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2006                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      sootProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

PAHs (A2 A3 A4);

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

ABF mechanism [16] will be used for this simulation. The chemistry and thermodynamic files of this mechanism are available in Chemkin format, and need to be converted to foam chemistry and thermodynamic files using `chemkinToFoam` command. The mechanism conversion might pose challenges of its own such as duplicate species and discontinuous temperature ranges leading to errors in using `chemkinToFoam` that sometimes are not easy to interpret. Since the new combustion model library is the focus of this work, the details of mechanism conversion are not covered here and only the final files are provided in the project files. The chemistry (reaction) and thermodynamic properties are `reactions.ABF` and `thermo.ABF-mod`, respectively, and they are placed in `constant` directory. `thermophysicalProperties` contains the path to chemistry and thermodynamic files under `foamChemistryFile`, and `foamChemistryThermoFile`, respectively.

`constant/thermophysicalProperties`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2006                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      thermophysicalProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

thermoType
{
    type            hePsiThermo;
    mixture         reactingMixture;
    transport       sutherland;
    thermo          janaf;
    energy          sensibleEnthalpy;
    equationOfState perfectGas;
    specie          specie;
}

inertSpecie N2;

chemistryReader foamChemistryReader;
foamChemistryFile "<constant>/reactions.ABF";
foamChemistryThermoFile "<constant>/thermo.ABF-mod";

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

4.3 Time Directory

4.3.1 Soot Fields

The first time directory, `0`, must contain `N_agg`, `N_pri`, `C_tot`, and `H_tot` files that `laminarSoot` reads to determine initial and boundary conditions. The type of boundary conditions for soot fields are similar to species, which is `zeroGradient` on walls. The initial values are computed for one spherical soot particle per kg of gas mixture, which 2nm in diameter (with 378 carbon and 20 hydrogen atoms), as

$$N^i_{agg} = N^i_{pri} = \frac{1}{Av} = 1.66054 \times 10^{-24} \frac{mol}{kg}. \tag{4.1}$$

$$C^i_{tot} = \frac{378}{Av} = 6.27684 \times 10^{-22} \frac{mol}{kg}. \tag{4.2}$$

$$H^i_{tot} = \frac{20}{Av} = 3.32108 \times 10^{-23} \frac{mol}{kg}. \tag{4.3}$$

The contents of `N_agg` is provided below. `N_pri`, `C_tot`, and `H_tot` files are similar except for the object name and initial value, so they are included in Appendix B.

`0/N_agg`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
```

```
|  \\    /   O peration     | Version:  v2006                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      N_agg;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform 1.66054e-24;

boundaryField
{
    sideWalls
    {
        type            zeroGradient;
    }
    upperWall
    {
        type            zeroGradient;
    }
    lowerWall
    {
        type            zeroGradient;
    }
    frontAndBack
    {
        type empty;
    }
}


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

4.3.2 Species

C2H4 and N2 contain the initial and boundary condition for mass fraction of C_2H_4 and N_2, respectively. zeroGradient condition is applied on walls. The initial value is 0.2 for C_2H_4, and 0.8 for N_2 consistent with the physics of the case explained in Section 4.1. The contents of $C2H4$ and $N2$ are given below.

0/C2H4

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2006                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      CH4;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

```

dimensions      [0 0 0 0 0 0 0];

internalField   uniform 0.2;

boundaryField
{
    sideWalls
    {
        type            zeroGradient;
    }
    upperWall
    {
        type            zeroGradient;
    }
    lowerWall
    {
        type            zeroGradient;
    }
    frontAndBack
    {
        type empty;
    }
}


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

0/N2

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2006                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      N2;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 0 0 0 0 0 0];
internalField   uniform 0.8;
boundaryField
{
    sideWalls
    {
        type            zeroGradient;
    }
    upperWall
    {
        type            zeroGradient;
    }
    lowerWall
    {
        type            zeroGradient;
    }
    frontAndBack
    {
        type empty;
    }
}
```

Table 4.1: The initial values and boundary conditions for temperature, pressure and velocity

Quantity	Filename	Initial Value	Boundary condition
Temperature	T	1800	zeroGradient
Velocity	U	(0 0 0)	zeroGradient
Pressure	p	1e5	zeroGradient
C_2H_4	C2H4	0.2	zeroGradient
N_2	N2	0.8	zeroGradient

The solver uses `Ydefault` as initial and boundary condition for all the rest of species that are not included in `0` directory. The boundary conditions are identical to C_2H_4 and N_2, but the initial value is `uniform 0`. `Ydefault` is included in Appendix B.

4.3.3 Temperature, Pressure, and Velocity

The initial values for `T` and `p` are `1800` and `1e5`, respectively. `U` is uniformly zero. The boundary condition is `zeroGradient` on walls for all three fields. The initial values and boundary conditions are listed in Table 4.1

4.4 System Directory

`controlDict` is modified to set `endTime` to 0.04s according to the residence time discussed in Section 4.1. The simulation time step, `deltaT`, is set to 10^{-5}s.

system/controlDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2006                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
application     reactingFoam;
startFrom       latestTime;
startTime       0;
stopAt          endTime;
endTime         0.04;
deltaT          1e-5;
writeControl    adjustable;
writeInterval   0.001;
purgeWrite      0;
writeFormat     ascii;
writePrecision  7;
writeCompression off;
timeFormat      general;
timePrecision   6;
runTimeModifiable true;
adjustTimeStep  no;
maxCo           0.1;
functions{
    #include "probes"
```

```
}
// ************************************************************************* //
```

`probe` object functions are included are the end of `controlDict` to collect data for validation with the benchmark code.

`system/probes`

```
probes
{
    type            probes;
    libs            (sampling);
    name            probes;
    writeControl    outputTime;
    writeInterval   1;
    fields          (N_agg N_pri C_tot H_tot);
    interpolationScheme cellPoint;
    probeLocations
    (
        ( 0.05 0.05 0.005 )
    );
    sampleOnExecute yes;
}
```

The convection terms in transport equations of soot fields are discretized using `limitedLinear` scheme. The soot fields are solved using `PBiCGStab` solver with `DILU` preconditioner and tolerance of 10^{-6} and relative tolerance of 0.1 (see `fvSchemes` and `fvSolution` in Appendix B).

4.5 Running and Post-processing

`Allrun` takes care of all running step including (i) sourcing OpenFOAM (ii) converting `reactor.msh` to OpenFOAM-readable format mesh (iii) decomposing the domain for six processor (iv) running `reactingFoam`, (iv) reconstructing the processor data, and (v) post processing using `postProcess.py` that reads benchmark data of EPTLSOOT and generates the plots and saves them in `figures` directory. As shown in Figure 4.2, the soot fields obtained by `laminarSoot` are in excellent agreement with the results of EPTLSOOT indicating the accuracy of the developed library. The mass fraction of key species involved in soot formation is shown Figure 4.3 that demonstrates a good agreement between OpenFOAM and EPTLSOOT results.

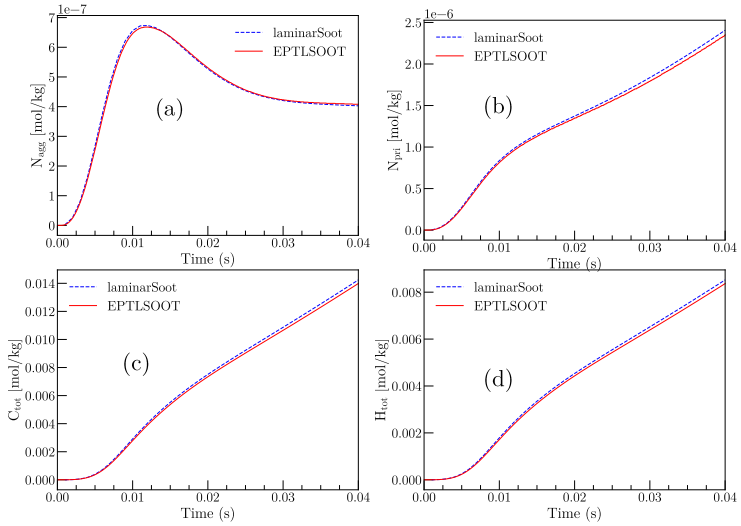

Figure 4.2: The comparison of (a) N_{agg} (b) N_{pri} (c) C_{tot} (c) H_{tot} obtained by `laminarSoot` with those of EPTLSOOT code

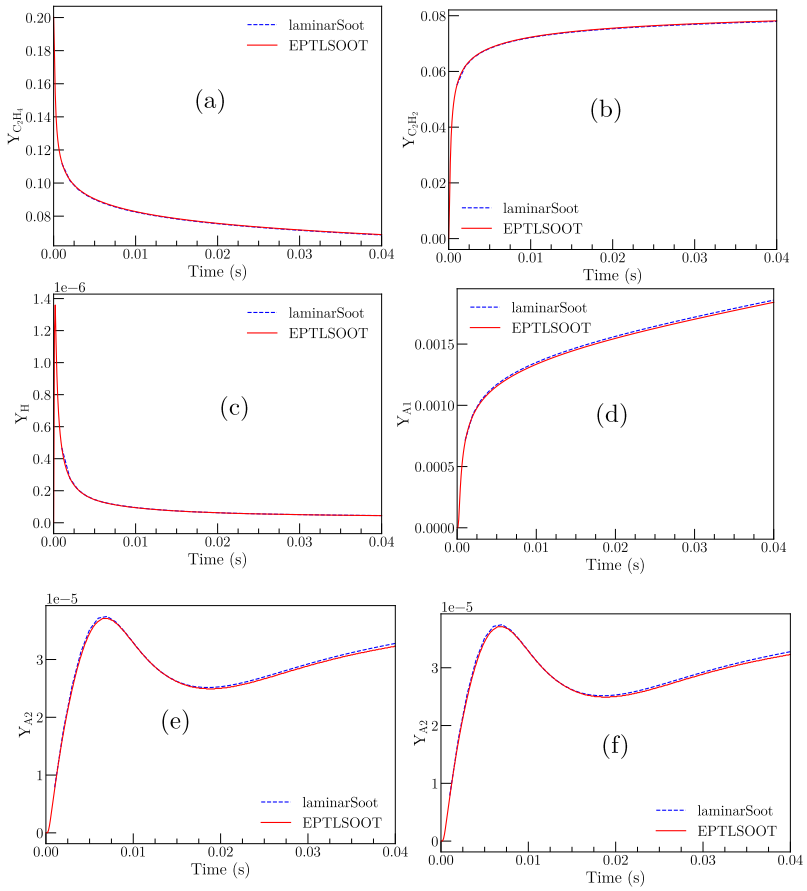

Figure 4.3: The comparison of mass fraction of a) C_2H_4 (b) C_2H_2 (c) H (c) A1 (d) A2 (e) A4 obtained by `laminarSoot` with those of EPTLSOOT code

Bibliography

[1] G. Myhre, D. Shindell, and J. Pongratz, "Anthropogenic and natural radiative forcing," 2014.

[2] "International carbon black association, carbon black user's guide," 2017.

[3] V. Palomares, A. Goñi, I. G. D. Muro, I. D. Meatza, M. Bengoechea, I. Cantero, and T. Rojo, "Conductive additive content balance in li-ion battery cathodes: Commercial carbon blacks vs. in situ carbon from lifepo4/c composites," *J. Power Sources*, vol. 195, pp. 7661–7668, 2010.

[4] H. Wang, "Formation of nascent soot and other condensed-phase materials in flames," *Proc Combust Inst.*, vol. 33, pp. 41–67, 2011.

[5] G. A. Kelesidis, E. Goudeli, and S. E. Pratsinis, "Flame synthesis of functional nanostructured materials and devices: Surface growth and aggregation," *Proc Combust Inst .*, vol. 36, pp. 29–50, 2017.

[6] A. Naseri, M. R. Kholghy, N. A. Juan, and M. J. Thomson, "Simulating yield and morphology of carbonaceous nanoparticles during fuel pyrolysis in laminar flow reactors enabled by reactive inception and aromatic adsorption," *Combustion and Flame*, vol. 237, p. 111721, 2022.

[7] M. R. Kholghy, A. Veshkini, and M. J. Thomson, "The core–shell internal nanostructure of soot–a criterion to model soot maturity," *Carbon*, vol. 100, pp. 508–536, 2016.

[8] M. R. Kholghy and G. A. Kelesidis, "Surface growth, coagulation and oxidation of soot by a monodisperse population balance model," *Combust. Flame*, 2021.

[9] P. T. Spicer, O. Chaoul, S. Tsantilis, and S. E. Pratsinis, "Titania formation by ticl4 gas phase oxidation, surface growth and coagulation," *J. Aerosol Sci.*, vol. 33, pp. 17–34, 2002.

[10] G. A. Kelesidis, E. Goudeli, and S. E. Pratsinis, "Morphology and mobility diameter of carbonaceous aerosols during agglomeration and surface growth," *Carbon*, vol. 121, pp. 527–535, 9 2017.

[11] E. Goudeli, M. L. Eggersdorfer, and S. E. Pratsinis, "Coagulation of agglomerates consisting of polydisperse primary particles," *Langmuir*, vol. 32, pp. 9276–9285, 2016.

[12] G. A. Kelesidis and S. E. Pratsinis, "Estimating the internal and surface oxidation of soot agglomerates," *Combust. Flame*, vol. 209, pp. 493–499, 2019.

[13] D. G. Goodwin, H. K. Moffat, and R. L. Speth, "Cantera: An object-oriented software toolkit for chemical kinetics, thermodynamics, and transport processes," *Caltech, Pasadena, CA*, vol. 124, 2009.

[14] F. Carbone, K. Gleason, and A. Gomez, "Probing gas-to-particle transition in a moderately sooting atmospheric pressure ethylene/air laminar premixed flame. part i: gas phase and soot ensemble characterization," *Combustion and Flame*, vol. 181, pp. 315–328, 2017.

[15] G. A. Kelesidis, E. Goudeli, and S. E. Pratsinis, "Morphology and mobility diameter of carbonaceous aerosols during agglomeration and surface growth," *Carbon*, vol. 121, pp. 527–535, 2017.

[16] J. Appel, H. Bockhorn, and M. Frenklach, "Kinetic modeling of soot formation with detailed chemistry and physics: laminar premixed flames of c2 hydrocarbons," *Combustion and flame*, vol. 121, no. 1-2, pp. 122–136, 2000.

[17] G. Blanquart and H. Pitsch, "Analyzing the effects of temperature on soot formation with a joint volume-surface-hydrogen model," *Combustion and Flame*, vol. 156, no. 8, pp. 1614–1626, 2009.

[18] M. R. Kholghy, G. A. Kelesidis, and S. E. Pratsinis, "Reactive polycyclic aromatic hydrocarbon dimerization drives soot nucleation," *Physical Chemistry Chemical Physics*, vol. 20, no. 16, pp. 10926–10938, 2018.

[19] K. Johansson, F. El Gabaly, P. Schrader, M. Campbell, and H. Michelsen, "Evolution of maturity levels of the particle surface and bulk during soot growth and oxidation in a flame," *Aerosol Science and Technology*, vol. 51, no. 12, pp. 1333–1344, 2017.

[20] H. Sabbah, L. Biennier, S. J. Klippenstein, I. R. Sims, and B. R. Rowe, "Exploring the role of pahs in the formation of soot: Pyrene dimerization," *The Journal of Physical Chemistry Letters*, vol. 1, no. 19, pp. 2962–2967, 2010.

[21] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

Study questions

1. What information are transferred from `laminar` combustion model to `reactingFoam` solver?
2. How reaction rates and heat release from chemical reactions in gas mixture are modified to consider the impact of soot formation?
3. How `laminarSoot` library is selected in a case?
4. What is the purpose of `correct()` function in the combustion model?

Appendix A

Appendix: Source Code

A.1 HACA-growth functions

laminarSoot.H

```
        // HACA Rate
        virtual tmp<volScalarField> k_4_HACA()
        {
            const volScalarField& T = this->thermo().T();
            dimensionedScalar oneKelvin(dimTemperature, scalar(1.0));
            dimensionedScalar k_unit(dimensionSet(0,3,-1,0,-1,0,0), scalar(1.0));
            return tmp<volScalarField>
            (
                new volScalarField
                (
                    "k_4_HACA",
                    (k_unit * 8.00e1 * pow(T/oneKelvin, 1.56) * exp(oneKelvin * (-1912.43) / T))
                )
            );
        }
        virtual tmp<volScalarField> k_5_HACA()
        {
            const volScalarField& T = this->thermo().T();
            dimensionedScalar oneKelvin(dimTemperature, scalar(1.0));
            dimensionedScalar k_unit(dimensionSet(0,3,-1,0,-1,0,0), scalar(1.0));
            return tmp<volScalarField>
            (
                new volScalarField
                (
                    "k_5_HACA",
                    (k_unit * 2.20e6 * exp(oneKelvin * (-3774.53) / T))
                )
            );
        }
        virtual tmp<volScalarField> k_6_HACA()
        {
            const volScalarField& T = this->thermo().T();
            dimensionedScalar oneKelvin(dimTemperature, scalar(1.0));
            dimensionedScalar k_unit(dimensionSet(0,3,-1,0,-1,0,0), scalar(1.0));
            return tmp<volScalarField>
            (
                new volScalarField
                (
                    "k_6_HACA",
                    (k_unit * 0.13e0 * pow(T/oneKelvin, 0.0))
                )
            );
        }
        virtual tmp<volScalarField> C_soot_0()
```

```
45          {
46              // Species indicies
47              label H_i = speciesIndicies_["H"];
48              label OH_i = speciesIndicies_["OH"];
49              label H2_i = speciesIndicies_["H2"];
50              label H2O_i = speciesIndicies_["H2O"];
51              label C2H2_i = speciesIndicies_["C2H2"];
52              label O2_i = speciesIndicies_["O2"];
53
54              const volScalarField& T = this->thermo().T();
55              volScalarField rho (this->thermo().rho());
56
57              dimensionedScalar oneKelvin(dimTemperature, scalar(1.0));
58              dimensionedScalar k_unit(dimensionSet(0,3,-1,0,-1,0,0), scalar(1.0));
59
60              // Forward Reaction Rates
61              // Units : m3/mol-s
62              volScalarField k_1(k_unit * 4.17e7 * exp(oneKelvin * (-6542.52) / T));
63              volScalarField k_2(k_unit * 1.00e4 * pow(T/oneKelvin, 0.734) * exp(oneKelvin * (-719.68) /
     T));
64              volScalarField k_3(k_unit * 2.00e7 * pow(T/oneKelvin, 0.0));
65
66              // Reverse Reaction Rates
67              // Units : m3/mol-s
68              volScalarField k_r_1(k_unit * 3.9e6 * exp(oneKelvin * (-5535.98) / T));
69              volScalarField k_r_2(k_unit * 3.68e2 * pow(T/oneKelvin, 1.139) * exp(oneKelvin *
    (-8605.94) / T));
70
71              // chi_soot_0
72              // dimensionedScalar chi_soot_0_denum_limit("chi_soot_0_denum_limit",dimless/dimTime,
    scalar(1e-20));
73              volScalarField chi_soot_0_denum
74              (
75                  max
76                  (
77                      k_r_1 * C(H2_i) + k_r_2 * C(H2O_i) + k_3 * C(H_i) + k_4_HACA() * C(C2H2_i) +
    k_5_HACA() * C(O2_i) + k_1 * C(H_i) + k_2 * C(OH_i),
78                      dimensionedScalar(dimless/dimTime, SMALL)
79                  )
80              );
81
82              dimensionedScalar chi_soot_CH(dimensionSet(0,-2,0,0,0,0,0), scalar(2.3e19));
83              //
84              volScalarField chi_soot_0
85              (
86                  (k_1 * C(H_i) + k_2 * C(OH_i)) / chi_soot_0_denum * chi_soot_CH
87              );
88
89              // C_soot_0
90              // unit: mol/m3
91              return tmp<volScalarField>
92              (
93                  new volScalarField
94                  (
95                      "C_soot_0",
96                      (A_tot_ / Av_ * chi_soot_0) * rho
97                  )
98              );
99          }
100
101         virtual tmp<volScalarField> alpha()
102         {
103             dimensionedScalar oneKelvin(dimTemperature, scalar(1.0));
104             dimensionedScalar alpha_min(dimless, scalar(0.0));
105             dimensionedScalar alpha_max(dimless, scalar(1.0));
106             const volScalarField& T = this->thermo().T();
107             return tmp<volScalarField>
108             (
```

```
109                 new volScalarField
110                 (
111                     min
112                     (
113                         max
114                         (
115                             tanh
116                             (
117                                 (12.56 - 0.00563 * T / oneKelvin) / log10 ( rho_soot_ * pi_ / 6.0 *
    pow(d_p_, 3.0) * Av_ / W_carbon_ ) -
118                                 1.38 + 0.00068 * T / oneKelvin
119                             ),
120                             alpha_min
121                         ),
122                         alpha_max
123                     )
124                 )
125             );
126         }
```

Appendix B

Appendix: Simulation case files

B.1 0 directory

0/N_pri

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2006                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      N_pri;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform 1.66054e-24;

boundaryField
{
    sideWalls
    {
        type            zeroGradient;
    }
    upperWall
    {
        type            zeroGradient;
    }
    lowerWall
    {
        type            zeroGradient;
    }
    frontAndBack
    {
        type empty;
    }
}


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

0/C_tot

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2006                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      C_tot;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform 6.27684e-22;

boundaryField
{
    sideWalls
    {
        type            zeroGradient;
    }
    upperWall
    {
        type            zeroGradient;
    }
    lowerWall
    {
        type            zeroGradient;
    }
    frontAndBack
    {
        type empty;
    }
}


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

0/H_tot

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2006                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      H_tot;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform 3.32108e-23;
```

```
21
22 boundaryField
23 {
24     sideWalls
25     {
26         type            zeroGradient;
27     }
28     upperWall
29     {
30         type            zeroGradient;
31     }
32     lowerWall
33     {
34         type            zeroGradient;
35     }
36     frontAndBack
37     {
38         type empty;
39     }
40 }
41
42
43 // ************************************************************************* //
```

0/Ydefault

```
1  /*--------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4  |  \\    /   O peration     | Version:  v2006                                 |
5  |   \\  /    A nd           | Website:  www.openfoam.com                      |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       volScalarField;
13     location    "0";
14     object      Ydefault;
15 }
16 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18 dimensions      [0 0 0 0 0 0 0];
19
20 internalField   uniform 0.0;
21
22 boundaryField
23 {
24     sideWalls
25     {
26         type            zeroGradient;
27     }
28     upperWall
29     {
30         type            zeroGradient;
31     }
32     lowerWall
33     {
34         type            zeroGradient;
35     }
36     frontAndBack
37     {
38         type empty;
39     }
40 }
41
42
```

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

B.2 system directory

system/fvSchemes

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2006                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
}

divSchemes
{
    default         none;

    div(phi,U)      Gauss limitedLinearV 1;
    div(phi,Yi_h)   Gauss limitedLinear 1;
    div(phi,K)      Gauss limitedLinear 1;
    div(phid,p)     Gauss limitedLinear 1;
    div(phi,epsilon) Gauss limitedLinear 1;
    div(phi,k) Gauss limitedLinear 1;
    div(phi,N_agg) Gauss limitedLinear 1;
    div(phi,N_pri) Gauss limitedLinear 1;
    div(phi,C_tot) Gauss limitedLinear 1;
    div(phi,H_tot) Gauss limitedLinear 1;
    div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear orthogonal;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         orthogonal;
}

```

```
60
61 // ************************************************************************* //
```

system/fvSolutions

```
1  /*--------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4  |  \\    /   O peration     | Version:  v2006                                 |
5  |   \\  /    A nd           | Website:  www.openfoam.com                      |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       dictionary;
13     location    "system";
14     object      fvSolution;
15 }
16 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18 solvers
19 {
20     "rho.*"
21     {
22         solver          diagonal;
23     }
24
25     p
26     {
27         solver           PCG;
28         preconditioner   DIC;
29         tolerance        1e-6;
30         relTol           0.1;
31     }
32
33     pFinal
34     {
35         $p;
36         tolerance        1e-6;
37         relTol           0.0;
38     }
39
40     "(U|h|k|epsilon|N_agg|N_pri|C_tot|H_tot)"
41     {
42         solver          PBiCGStab;
43         preconditioner  DILU;
44         tolerance       1e-6;
45         relTol          0.1;
46     }
47
48     "(U|h|k|epsilon|N_agg|N_pri|C_tot|H_tot)Final"
49     {
50         $U;
51         relTol          0;
52     }
53
54     Yi
55     {
56         $hFinal;
57     }
58 }
59
60 PIMPLE
61 {
62     momentumPredictor no;
63     nOuterCorrectors  1;
```

```
64      nCorrectors     2;
65      nNonOrthogonalCorrectors 0;
66  }
67
68  // ************************************************************************* //
```