Cite as: Kazemi, S.: Implementation of a Sectional Population Balance Model (SPBM) in laminar combustion model. In Proceedings of CFD with OpenSource Software, 2023, Edited by Nilsson. H.,
http://dx.doi.org/10.17196/OS_CFD#YEAR_2023

CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

Implementation of a Sectional Population Balance Model (SPBM) in laminar combustion model

Developed for OpenFOAM-v2112

Author:
Sina Kazemi
Carleton University
sinakazemi@cmail.carleton.ca

Peer reviewed by:
Dr. Reza Kholghy
Johannes Hansson

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 14, 2024

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 laminarSPBM combustion model with reactingFoam
- How to configure a case to employ SPBM aerosol model for simulating the formation and evolution of soot particles

The theory of it:

- How Sectional Population Balance Models (SPBM) simulates soot formation and evolution
- How SPMB employs source/sink terms to provide Particle Size Distribution

How it is implemented:

- How to define sections to consider the particle size distribution
- How to make the chemical source term section-specific
- How to model collision between particles from different sizes

How to modify it:

- How to distribute chemical source terms between sections to model soot evolution in each section

Prerequisites

This is an example for the prerequisites. The reader is expected to know the following in order to get maximum benefit out of this report:

- Basic knowledge of fluid mechanics, combustive systems, chemistry, and soot formation process
- Fundamentals of computational methods for fluid dynamics, chemistry solvers, and solving transport equations
- Basic knowledge of OpenFOAM solvers and libraries and how to modify them
- A good familiarity with C++ programming language
- Some experiences in using reactive simulation cases in OpenFOAM

Contents

Nomenclature

Acronyms

DEM	Discrete Element Modelling
HACA	Hydrogen abstraction carbon addition
MPBM	Monodisperse Population Balance Model
PAH	Polycyclic Aromatic Hydrocarbons
PSD	Particle Size Distribution
SPBM	Sectional Population Balance Model

English symbols

Av	Avogadro's number	$6.02214076 \times 10^{23}$ 1/mol
C_{agg}	carbon content of soot agglomerates	mol/kg
C_{tot}	total carbon content of soot particles	mol/kg
D	diffusion coefficient	m^2/s
d	diameter	m
d_g	gyration diameter	m
d_m	mobility diameter	m
d_p	primary particle diameter	m
I	soot formation/destruction rate	mol/kg − s
k	reaction rate constant	m^3/mol − s
k_B	Boltzmann constant	$1.3806488 \times 10^{-23}$ m^2kg/s^2−K
m	mass	kg
MS	number of sections	
n_p	number of primary particles	
N_{agg}	number density of agglomerates	mol/kg
N_{pri}	number density of primary particles	mol/kg
ROP	rate of production	mol/m^3−s
S	source term	mol/kg − s
SF	progression factor	
T	temperature	K
t	time	s
U	number of carbon atoms in each agglomerate	mol
u	velocity	m/s
W	molecular weight	kg
f	friction factor	kg − m/s^2

Greek symbols

β	collision frequency	m^3/s
δ	Kronecker delta	
λ	mean free path	m
μ	gas dynamic viscosity	kg/m-s
ρ	gas density	kg/m^3

ρ_{soot}	soot density	$\mathrm{kg/m^3}$
ξ	coagulation efficiency	

Subscripts

agg	agglomerate
carbon	carbon material
co	coagulation
g	gyration
gas	gas phase
inc	inception
m	mobility
ox	oxidation
pri	primary particle
sg	surface growth

Chapter 1

Introduction

1.1 Background

Each year, approximately 9.5 million tons of soot are released into the environment, ranking it as the third-largest contributor to climate change, following carbon dioxide and methane. Carbon Black, having an identical chemical and physical structure as soot, stands as the most significant flame-made nanoparticle in terms of both worth and quantity. Annually, approximately 15 million tons of Carbon Black, valued at 17 billion USD [1], are manufactured and employed across a range of industries. Its applications span from reinforcing rubber products (notably tires) [2] to uses in paint, toner [3], and lithium-ion batteries [4]. The environmental impacts of soot and the extensive industrial applications of Carbon Black have rendered studying these carbonaceous nanoparticles highly appealing, attracting researchers from various disciplines in recent years [5].

Carbon Black formation is a complex process characterized by multiple steps with different time and length scales [6]. The formation process starts with the inception of particles from the gas or vapor phase [7], followed by mass growth through particle surface reactions [8, 9] and coalescence [10]. In the meantime, particles collide, forming fractal-like particles conventionally called agglomerates. Under specific conditions, the presence of oxidative agents can lead to mass reduction in Carbon Black particles due to oxidation, and in some cases, even fragmentation can occur [11]. These various steps in the Carbon Black formation pathway govern the process yield, Carbon Black composition (i.e, particle carbon to hydrogen ratio), and the morphology of the particles, including aspects like polydispersity, primary particle diameter, specific surface area, effective density, etc [12, 13].

Researchers have utilized various modeling approaches to simulate the formation and evolution of Carbon Black under different pyrolysis conditions. These models constitute a wide spectrum, from the computationally efficient Monodisperse Population Balance Model (MPBM) [14] to the most accurate but computationally demanding one, Discrete Element Models (DEM) [15]. MPBM is capable of accurately simulating the evolution of particles while maintaining low computational costs, particularly when the particles have achieved a self-preserving morphology and size distribution [16]. Nevertheless, a significant drawback of the MPBM is its failure to incorporate the particle size distribution (PSD) which can lead to loss of accuracy in some cases [17]. Sitting between the two extremes is the Sectional Population Balance Model (SPBM), which has garnered attention within the soot modeling community. The SPBM is known for providing acceptable accuracy in simulating soot formation processes while maintaining a computationally affordable approach [18]. Additionally, its ease of implementation makes it a practical choice for researchers working in the field of soot modeling. The SPBM follows a similar formulation as the MPBM, but it addresses the most significant weakness of the MPBM by incorporating particle size distribution (PSD) predictions [19].

1.2 Laminar Combustion Model

The `reactingFoam` solver employs combustion models to calculate the rate of production/destruction of the chemical species, providing the source/sink terms in the transport equations of mass fractions, and the heat released/adsorbed by the chemical reactions, which appear as source/sink terms in the enthalpy (`he`) equation. Previously, the `laminar` combustion model (which neglects the effect of turbulence on the reaction rates, `R`, and the heat released/adsorbed by the chemical reactions, `Qdot`) was extended to model soot formation in the Eulerian framework. The implemented soot model in `laminarSoot` was based on the Monodisperse Population Balance model (MPBM) [20]. This report aims to further develop `laminarSoot` to consider particle polydispersity by implementing the Sectional Population Balance Model (SPBM).

1.3 Objectives

This project aims to enhance the capabilities of the existing library, `laminarSoot`, which was originally built upon the `laminar` combustion model for simulating soot formation with the Monodispersed Population Balance Model (MPBM). The objective involves the incorporation of the Sectional Population Balance Model (SPMB), achieved through

- Defining new fields for doing calculations for each particle size based on the particle sections
- Modifying existing methods that are responsible for calculating source terms to make them size-dependent
- Implementing new methods to re-distribute source terms between particle sections
- Implementing new methods to model particle agglomeration through collision between different sections
- Modifying the set of the transport equations to solve them for each section

At the end, a simple zero-dimensional case will be set up to investigate the performance of the `laminarSPBM` library.

Chapter 2

Theory

This chapter begins with an introduction to the properties of soot particles and the concept of population balance models, aiming to familiarize the reader with the approach to modeling soot particle formation and evolution within the Eulerian framework. Subsequently, a comprehensive overview of the theoretical background of the Sectional Population Balance Model (SPMB) will be presented. For additional insights into the chemical and physical mechanisms contributing to soot formation, the reader is directed to relevant sections in the appendix to ensure the report's self-sufficiency.

2.1 Soot particle

Soot agglomerates are composed of spherical carbonaceous particles attached by either chemical or physical bonds. The fractal-like structure of these agglomerates is characterized by key parameters such as the number of primary particles, denoted as n_p, as well as the diameters of primary particles, d_pri, mobility diameter, d_m, and gyration diameter, d_g. Figure 2.1 depicts a schematic of different soot diameters on a soot agglomerate consisting of 16 primary particles. The following outlines the definitions of these parameters and provides the corresponding formulas for their calculation.

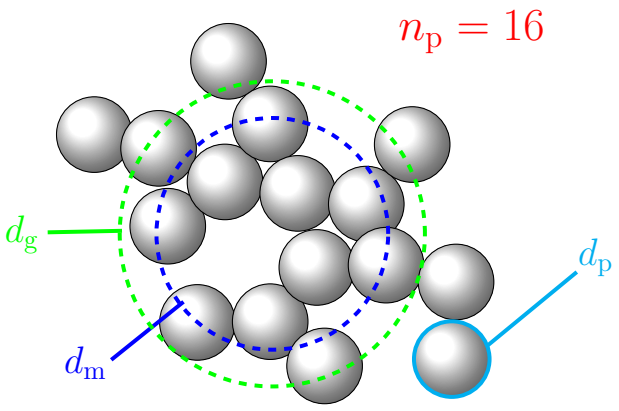

Figure 2.1: Schematic of a soot agglomerate with $n_\mathrm{p} = 16$ spherical primary particles

For each agglomerate, n_p is the number of the primary particles which can be easily calculated by

$$n_\mathrm{p} = \frac{N_\mathrm{pri}}{N_\mathrm{agg}}. \tag{2.1}$$

The diameter of primary particles is assumed to be identical in each agglomerate and can be determined based on the total carbon content of the agglomerate using

$$d_{\mathrm{p}} = \left(\frac{6}{\pi} \frac{C_{\mathrm{agg}} \cdot W_{\mathrm{carbon}}}{\rho_{\mathrm{soot}}} \frac{1}{N_{\mathrm{pri}} \cdot Av} \right)^{1/3}. \tag{2.2}$$

The mobility diameter of a soot agglomerate is the diameter of a sphere exhibiting the same translational behaviors as the agglomerate and it is given by

$$d_{\mathrm{m}} = d_{\mathrm{p}} \cdot n_{\mathrm{p}}^{0.45}. \tag{2.3}$$

Similarly, the gyration diameter of a soot agglomerate is the diameter of a sphere with the same rotational properties as the agglomerate and it is determined by

$$d_{\mathrm{g}} = \begin{cases} d_{\mathrm{m}}/n_{\mathrm{p}}^{-0.2} + 0.4 & \text{if } n_{\mathrm{p}} > 1.5 \\ d_{\mathrm{m}}/1.29 & \text{if } n_{\mathrm{p}} \leq 1.5 \end{cases}. \tag{2.4}$$

Finally, the total carbon, C_{tot}, can be derived based on the number density of agglomerates, N_{agg}, and the carbon content of each agglomerate, C_{agg}, using

$$C_{\mathrm{tot}} = N_{\mathrm{agg}} Av C_{\mathrm{agg}}. \tag{2.5}$$

2.2 Transport equations for soot agglomerates

Population balance models adopt the Eulerian description of particles, wherein specific physical properties representative of the particle population—such as number densities, total carbon, or surface area—are treated as continuous quantities. These properties are described by solving scalar transport equations. The soot model utilized in this project adheres to the SPBM framework which is based on the number density of primary particles, N_{pri}, number density of agglomerates, N_{agg}, and total carbon content, C_{tot} [21]. These quantities can be extended as needed for specific applications. However, in order to maintain computational cost affordable, this project focuses on tracking only three aforementioned variables using two transport equations

$$\frac{\partial}{\partial t}\left(\rho N_{\mathrm{agg}}^{i}\right) + \nabla \cdot \left(\rho u N_{\mathrm{agg}}^{i}\right) + \nabla^{2}\left(\rho D N_{\mathrm{agg}}^{i}\right) = \rho\left(S_{\mathrm{agg}}^{i}\right) \tag{2.6}$$

$$\frac{\partial}{\partial t}\left(\rho N_{\mathrm{pri}}^{i}\right) + \nabla \cdot \left(\rho u N_{\mathrm{pri}}^{i}\right) + \nabla^{2}\left(\rho D N_{\mathrm{agg}}^{i}\right) = \rho\left(S_{\mathrm{pri}}^{i}\right). \tag{2.7}$$

Where superscript i denotes the particle size section, and source term, S, is the total source term that is determined by various chemical processes. The way that the source terms are calculated will be discussed in Sections 2.4–2.5. It is noteworthy that in SPBM the total carbon content, C_{tot}^{i}, is indirectly determined after obtaining particle number densities.

The diffusion coefficient of soot particle, D, used in Eqs. (2.6)–(2.7) is calculated as

$$D = \frac{k_B T}{f}, \tag{2.8}$$

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

$$f = \frac{3\pi\mu d_{\mathrm{m}}^{i}}{C}, \tag{2.9}$$

$$C = 1 + \frac{2\lambda}{d_{\mathrm{m}}^{i}}\left(1.21 + 0.4\exp\left(\frac{-0.78 d_{\mathrm{m}}^{i}}{\lambda}\right)\right), \tag{2.10}$$

where λ is the mean free path of gas given as

$$\lambda = \frac{\mu}{\rho}\sqrt{\frac{\pi W_{\mathrm{gas}}}{2 k_B Av T}}. \tag{2.11}$$

2.3 Monodisperse Population Balance Model (MPBM)

The main simplification of Monodisperse Population Balance Models is the omission of polydispersity in soot particles. Thus, for simulating soot formation and evolution, at each time step, all transport equations (refer to Section 2.2) are solved only for one particle size. Subsequently, all soot particle properties (including mass, size, etc.) are updated, and the computations in the next step are performed using the updated properties. The primary advantage of MPBM is to keep computational costs low by solving all transport equations only once in each time step, but in many circumstances, this can lead to inaccurate results.

2.4 Sectional Population Balance Model (SPBM)

This section aims to present the formulation required to implement a two-equation sectional population balance model using the class methods in `laminarSoot` library that calculate chemical source terms (refer to Appendix A for more details about source term calculations). Therefore, it is assumed that the models that determine the contribution of different mechanisms (e.g., inception, coagulation, surface growth by HACA or PAHs, and fragmentation.) in the soot formation process are available and can be employed in the new soot library–`laminarSPBM`–with some modifications.

Despite MPBM, SPBM considers particle polydispersity, which means particles are divided into different fixed sections based on their size. In the SPBM scheme, soot particles transfer between different sections as they gain or lose mass during their evolution. As a result, at each moment, there are soot particles with different sizes, each exhibiting different behaviors as many of the derived models are size-dependent.

It is assumed that soot particles will be distributed among MS size sections as they evolve. The first section is assigned to soot particles with specific moles of carbon atoms, while the other sections are defined using an arbitrary fixed geometric progression factor. Therefore, if the first bin has U_1 number of moles of carbon atoms and the progression factor is SF, the number of moles of carbon atoms of the i^{th} section can be calculated by

$$U_i = U_1 SF^{i-1}. \tag{2.12}$$

The schematic of section distribution is illustrated in figure 2.2.

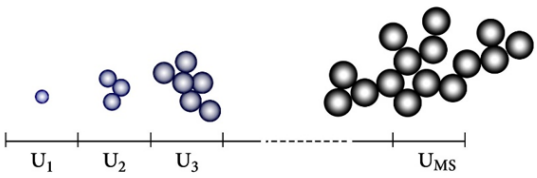

Figure 2.2: Schematic of soot particle sections

Evidently, for each variable tracked during the simulation, there is a conservation equation for each section. Therefore, for the tracking number density of primary particles, N^i_{pri}, the number density of agglomerates, N^i_{agg}, and total number of carbon atoms C^i_{tot} there would be $2MS$ conservation equations to solve. To be more specific, two conservation equations should be solved for N^i_{pri} and N^i_{agg}. Then the total number of carbon atoms would be obtained by having the size of each bin and the number of particles in it ($C^i_{\text{tot}} = N^i_{\text{agg}} U_i$). It is important to note that the conservation equations for each section in SPBM are identical to the one implemented in `laminarSoot`. However, overall source terms may vary due to their size-dependent nature (e.g., HACA) or because the phenomenon is more complex in a polydisperse system (e.g., agglomeration). In what follows, the source terms that should be used in SPBM will be explained.

2.4.1 Inception

The inception process describes the birth of the incipient particles from the chemical species in the gas phase. Inception occurs in exactly the same way as MPBM and only affects the variables in the first section. Therefore, the number of moles of carbon added to the first section can be calculated by

$$I_{\text{inc}} = \sum C_{ij}[1]\frac{ROP_{ij}}{\rho_{\text{gas}}} = S_{\text{C}_{\text{tot,inception}}}, \tag{2.13}$$

where C_{ij} is the total number of carbons in i^{th} and j^{th} PAHs and ROP_{ij} is the rate of production of chemically bonded dimers. Note that $S_{\text{C}_{\text{tot,inception}}}$ is identical to the source term calculated in `laminarSoot` as the inception process does not depend on the size of soot particles. For more information about the inception source term please refer to Section A.1. The number of particles added to the first section would be

$$S^1_{\text{inc}} = \left(\frac{\partial N^1_{\text{pri}}}{\partial t}\right)_{\text{inc}} = \left(\frac{\partial N^1_{\text{agg}}}{\partial t}\right)_{\text{inc}} = \frac{I_{inc}}{U_1}\frac{1}{Av}. \tag{2.14}$$

Therefore, all equations should be solved like MPBM for the first section. Then, the number of added particles is calculated using Eq. (2.14).

2.4.2 Surface growth

Various pathways can be considered for the surface growth mechanism, such as PAH addition and HACA. The surface growth processes do not affect the total number of agglomerates or primary particles, but they alter the number of agglomerates and primary particles in each section by moving them into bins with higher mass. Again, the source term calculated in `laminarSoot` can be employed in `laminarSPBM` after some modifications, as they depend on the particle properties

$$I_{\text{sg},i} = S_{C_{\text{tot,growth},i}} = \left(\left(\frac{\partial C_{\text{tot}}}{\partial t}\right)^i_{\text{HACA}} + \left(\frac{\partial C_{\text{tot}}}{\partial t}\right)^i_{\text{PAH}}\right). \tag{2.15}$$

Where $i \in \{1, 2, 3, \ldots, MS\}$. The first term on the right-hand side of Eq. (2.15) is the source term due to the HACA surface growth process and is calculated by Eq. (A.16). The latter one is the contribution of PAH adsorption and determined by Eq. (A.32). The mass gained by surface growth at each section should be transferred to the next bins as the mass of the bins is constant. For the number density of agglomerate, determining source terms is done by

$$S^i_{\text{agg,sg}} = \frac{1}{Av}\begin{cases} -\frac{I_{\text{sg},1}}{U_2-U_1}, & \text{if } i = 1 \\ \frac{I_{\text{sg},i-1}}{U_i-U_{i-1}} - \frac{I_{\text{sg},i}}{U_{i+1}-U_i}, & \text{if } i = 2, \ldots, MS-1 \\ \frac{I_{\text{sg},MS-1}}{U_{MS}-U_{MS-1}}, & \text{if } i = MS \end{cases}. \tag{2.16}$$

For the number density of primary particles, source terms are given by

$$S^i_{\text{pri,sg}} = \frac{1}{Av}\begin{cases} -\frac{I_{\text{sg},1}}{U_2-U_1}, & \text{if } i = 1 \\ \frac{I_{\text{sg},i-1}}{U_i-U_{i-1}}n_{\text{p},i-1} - \frac{I_{\text{sg},i}}{U_{i+1}-U_i}n_{\text{p},i}, & \text{if } i = 2, \ldots, MS-1 \\ \frac{I_{\text{sg},MS-1}}{U_{MS}-U_{MS-1}}n_{\text{p},i-1}, & \text{if } i = MS \end{cases}. \tag{2.17}$$

In the above expression, $n_{\text{p}} = N_{\text{pri}}/N_{\text{agg}}$ is the number of primary particles per agglomerate.

2.4.3 Oxidation

Despite surface growth which increases the mass of particles and pushes them to bins representing larger particles, the oxidation process partially destroys particles and transfers them to the previous sections. The rate of oxidation is available from `laminarSoot` based on Eq. (A.36). Thus, the mass destruction due to oxidation would be

$$I_{\mathrm{ox},i} = S_{C_{\mathrm{tot,ox},i}} = \left(\frac{\partial C_{\mathrm{tot}}}{\partial t}\right)^{i}_{\mathrm{ox}} . \tag{2.18}$$

Again, as the mass of each section is fixed, the effect of the oxidation on the particle size distribution is given by

$$S^{i}_{\mathrm{agg,ox}} = \frac{1}{Av}\begin{cases} \frac{I_{\mathrm{ox},2}}{U_2-U_1} - \frac{I_{\mathrm{ox},1}}{U_1}, & \text{if } i = 1 \\ \frac{I_{\mathrm{ox},i+1}}{U_{i+1}-U_i} - \frac{I_{\mathrm{ox},i}}{U_i-U_{i-1}}, & \text{if } i = 2,\ldots,MS-1 \\ \frac{I_{\mathrm{ox},MS}}{U_{MS-1}-U_{MS}} n_{\mathrm{p},i-1}, & \text{if } i = MS \end{cases} . \tag{2.19}$$

Similarly, for the number density of primary particles, the source terms are

$$S^{i}_{\mathrm{pri,ox}} = \frac{1}{Av}\begin{cases} \frac{I_{\mathrm{ox},2}}{U_2-U_1} n_{\mathrm{p},2} - \frac{I_{\mathrm{ox},1}}{U_1}, & \text{if } i = 1 \\ \frac{I_{\mathrm{ox},i+1}}{U_{i+1}-U_i} n_{\mathrm{p},i+1} - \frac{I_{\mathrm{ox},i}}{U_i-U_{i-1}} n_{\mathrm{p},i}, & \text{if } i = 2,\ldots,MS-1 \\ \frac{I_{\mathrm{ox},MS}}{U_{MS-1}-U_{MS}} n_{\mathrm{p},MS}, & \text{if } i = MS \end{cases} . \tag{2.20}$$

2.4.4 Coagulation

Coagulation is the process during which solid and hard soot particles collide and attach at the point of contact leading to larger agglomerates. The contribution of coagulation in SPBM differs significantly from what occurs in MPBM, as particles from various size categories can coagulate in a polydisperse system. It should be noted that coagulation increases the size of an agglomerate and moves it into the upper sections while reducing the total number of agglomerates. Additionally, coagulation does not affect the total number of primary particles, but it alters the primary particle and agglomerate number density in each section. The following algorithm takes care of the source terms

$$S^{i}_{\mathrm{agg,co}} = Av\rho_{\mathrm{gas}}\Big(\sum_{j}\sum_{k}\left(1-\frac{\delta_{jk}}{2}\right)\eta_{ijk}\beta_{jk}\xi_{jk}N^{j}_{\mathrm{agg}}N^{k}_{\mathrm{agg}} - N^{i}_{\mathrm{agg}}\sum_{m=1}^{MS}\beta_{im}\xi_{im}N^{m}_{\mathrm{agg}}\Big) \tag{2.21}$$

$$S^{i}_{\mathrm{pri,co}} = Av\rho_{\mathrm{gas}}\Big(\sum_{j}\sum_{k}\left(1-\frac{\delta_{jk}}{2}\right)\eta_{\mathrm{p},ijk}\eta_{ijk}\beta_{jk}\xi_{jk}N^{j}_{\mathrm{agg}}N^{k}_{\mathrm{agg}} - N^{i}_{\mathrm{pri}}\sum_{m=1}^{MS}\beta_{im}\xi_{im}N^{m}_{\mathrm{agg}}\Big) \tag{2.22}$$

$$\{\forall k \in [1,i] \wedge j \in [k,i] \mid U_{i-1} < U_j + U_k < U_{i+1}\}.$$

Where δ_{jk} is Kronecker delta, β_{jk} and ξ_{jk} are collision kernel and coagulation efficiency of two agglomerates from the j^{th} and the k^{th}. In this report, it is assumed that the collision kernel, β_{jk}, has a constant value. For each pair of sections, $\eta_{p,ijk}$ is given by

$$\eta_{p,ijk} = \frac{U_i}{U_j + U_k}\left(n_{\mathrm{p},j} + n_{\mathrm{p},k}\right). \tag{2.23}$$

The newly formed agglomerate should be transferred into two consecutive sections. That is the reason for including η_{ijk} in the source term formulas above. The term η_{ijk} is calculated by

$$\eta_{ijk} = \begin{cases} \frac{U_{i+1}-(U_j+U_k)}{U_{i+1}-U_i}, & \text{if } U_i \leq U_j + U_k < U_{i+1} \\ \frac{U_{i-1}-(U_j+U_k)}{U_{i-1}-U_i}, & \text{if } U_{i-1} < U_j + U_k < U_i \\ 0, & \text{else} \end{cases} \tag{2.24}$$

2.5 Overall source terms

The source terms on the right-hand side of Equations (2.6) and (2.7) are determined by summing the source terms of each mechanism.

$$S^i_{\text{agg}} = S^1_{\text{agg,inc}} + S^i_{\text{agg,sg}} + S^i_{\text{agg,ox}} + S^i_{\text{agg,co}} \tag{2.25}$$

$$S^i_{\text{pri}} = S^1_{\text{pri,inc}} + S^i_{\text{pri,sg}} + S^i_{\text{pri,ox}} + S^i_{\text{pri,co}} \tag{2.26}$$

2.6 Total properties

Intensive properties of the soot particles, such as d_m, d_g, n_p, etc., should be calculated by performing arithmetic averaging over all soot particles in all sections using

$$X_{\text{tot}} = \frac{\sum_{\text{sections}} X^i N^i_{\text{agg}}}{\sum_{\text{sections}} N^i_{\text{agg}}}. \tag{2.27}$$

Extensive properties, like total carbon content, total surface area, etc., are computed through summation across all sections by

$$X_{\text{tot}} = \sum_{\text{sections}} X^i N^i_{agg}. \tag{2.28}$$

Chapter 3

Implementation

3.1 Overview

This chapter aims to describe the transformation of the `laminarSoot` combustion model, previously developed for modeling soot formation in a monodisperse manner [20], into a new combustion model named `laminarSPBM` which adopts a sectional aerosol modeling approach. This report primarily focuses on the modifications required in different parts of the existing code, providing minimal explanations about the role of each part. Therefore, it is highly recommended that readers first familiarize themselves with the fundamentals of soot modeling by referring to the documentation of the `laminarSoot`[1] model and understand how different parts of that library work together to simulate various soot formation processes.

The sections in this chapter are ordered to enable readers to easily follow and comprehend the necessary modifications. Explanations are provided in some places to clarify why certain modifications are unnecessary, aiding readers in understanding the procedure. However, for a comprehensive understanding of the modifications and the ability to replicate them, readers should refer to the publicly available code provided in the supplementary materials.

It is important to note that `laminarSPBM` has been tested, compiled, and run without any issues at the time of writing this report. Nevertheless, it may encounter problems over time, as the OpenFOAM package itself may undergo variations.

3.2 Creating the base library

Here, the implementation of `laminarSPBM` is carried out based on the existing combustion model called `laminarSoot`, which was previously developed using the `laminar` combustion model [20]. Therefore, the implementation starts with copying and renaming `laminarSoot`. After downloading the project file of `laminarSoot` into any directory, one should open a terminal within that directory and execute the following commands.

Instruction for copying the new library

```
foam
cp -r ./laminarSoot $WM_PROJECT_USER_DIR/src/laminarSPBM
cd $WM_PROJECT_USER_DIR/src/laminarSPBM
mv laminarSoot.H laminarSPBM.H
mv laminarSoot.C laminarSPBM.C
mv laminarSoots.C laminarSPBMs.C
sed -i s/"laminarSoot"/"laminarSPBM"/g *.*
```

There is no need to modify the `Make/options` file as it already includes all the required libraries. However, the `Make/files` file should be updated to include the following content.

[1] Implementation of a Monodisperse Population Balance Model in laminar combustion model.

Make/files

```
laminarSPBMs.C

LIB = $(FOAM_USER_LIBBIN)/liblaminarSPBM
```

3.3 Particle size sections

The particle size sections should be defined before starting the simulation, and based on Eq. (2.12) two inputs are required (the number of sections and the progression factor) from the user. These two inputs are received in `sootProperties` file through the following code:

laminarSPBM.C

```
    n_secs_
    (
        "numberOfSections",
        dimensionSet(0,0,0,0,0,0,0),
        sootProps_
    ),
    spacing_
    (
        "spacingFactor",
        dimensionSet(0,0,0,0,0,0,0),
        sootProps_
    ),
```

Two additional functions are called in the constructor to create the fields and size sections for the sectional model.

laminarSPBM.C

```
    if (integrateReactionRate_)
    {
        Info<< "    using integrated reaction rate" << endl;
    }
    else
    {
        Info<< "    using instantaneous reaction rate" << endl;
    }

    create_fields();
    build_C_agg_sec();
    createPAHProps();
    createDimerProps();
    createSpeciesProps();
```

Note that, these two functions must be declared in the header file like other functions that are called in the class constructor.

laminarSPBM.H

```
        virtual void create_fields();
        virtual void build_C_agg_sec();

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

3.4 Tracked and Derived Fields

For all sections, we declare and initialize the required fields. These fields include fields for three tracked variables, N^i_{pri}, N^i_{agg}, and C^i_{tot}, as well as fields for source terms plus fields needed for source

terms calculations. Since solving transport equations in SPBM necessitates a list of `volScalarField` for each variable, the type of declared data is `PtrList<volScalarField>`. The only exception is the inception source terms that only impact the first section of the particle size, so their type is `volScalarField`. Note that in the `laminarSoot`, the type of variables are `volScalarField` because only one field is needed for each variable.

laminarSPBM.H

```
113     // Soot tracked fields - sectional
114     PtrList<volScalarField> N_agg_sec_;
115     PtrList<volScalarField> N_pri_sec_;
116     PtrList<volScalarField> C_tot_sec_;
117
118     // Morphology - sectional
119     PtrList<volScalarField> n_p_sec_;
120     PtrList<volScalarField> m_agg_sec_;
121     PtrList<volScalarField> d_p_sec_;
122     PtrList<volScalarField> d_m_sec_;
123     PtrList<volScalarField> d_g_sec_;
124     PtrList<volScalarField> A_tot_sec_;
125
126     // Source Terms - sectional
127     // Inception
128     volScalarField I_inc_C_tot_;
129     // Surface growth
130     PtrList<volScalarField> I_grow_C_tot_sec_;
131     // Oxidation
132     PtrList<volScalarField> I_ox_C_tot_sec_;
133     // Coagulation
134     PtrList<volScalarField> I_coag_N_agg_sec_;
135     PtrList<volScalarField> I_coag_N_pri_sec_;
136     volScalarField sum1_inside_; //for coagulation calculations
137     volScalarField sum2_all_; //for coagulation calculations
138
139     // Source terms
140     PtrList<volScalarField> S_N_agg_sec_;
141     PtrList<volScalarField> S_N_pri_sec_;
142
143     // Total variables
144     volScalarField N_agg_;
145     volScalarField N_pri_;
146     volScalarField C_tot_;
147     volScalarField d_m_;
```

The initialization fields are included in a separate file named `createSectionalFields.H`. Because the number of sections is specified by the user, and its value is usually greater than 25, it is not feasible to read the initial values from the zero time directory. Therefore, the keyword `NO_READ` is used in the initialization and the initial value is specified in the code. To have a list of fields for each variable `forAll` command is used, and the number of iterations is specified by the number of sections, `secNum`.

The important point to note is that for variables solved by transport equations, N_{agg} and N_{pri}, one should specify boundary conditions. However, due to the large number of fields, which is usually unknown, reading boundary conditions from the time directories is not feasible. The solution for this is to group the field for all sections together. In these situations, the solver refers to the default boundary condition if the field file is not located in the time directory. The code below defines the variable `N_agg_sec` for tracking N_{agg} in each section, as well as `C_agg_sec` for the section creation.

createSectionalFields.H

```
39     tmp<volScalarField> tNPridefault;
40     tmp<volScalarField> tNAggdefault;
41
42     secNum_ = 0;
43
44     forAll(secNum_, sec)
```

```
45      {
46
47        C_agg_sec_.set
48            (
49                sec,
50                new volScalarField
51                (
52                    IOobject
53                    (
54                        "C_agg_sec" + std::to_string(sec),
55                        this->mesh().time().timeName(),
56                        this->mesh(),
57                        IOobject::NO_READ,
58                        IOobject::AUTO_WRITE
59                    ),
60                    this->mesh(),dimensionedScalar("C_agg_sec" + std::to_string(sec), dimensionSet
      (0,0,0,0,1,0,0),0.0)
61                )
62            );
63
64            IOobject time0AggIO
65            (
66                   IOobject::groupName("N_agg", ""),
67                   this->mesh().time().timeName(0),
68                   this->mesh(),
69                   IOobject::MUST_READ,
70                   IOobject::NO_WRITE
71            );
72
73            tNAggdefault = new volScalarField(time0AggIO, this->mesh());
74
75            N_agg_sec_.set
76            (
77                sec,
78                new volScalarField
79                (
80                    IOobject
81                    (
82                        IOobject::groupName( "N_agg_sec" + std::to_string(sec), ""),
83                        this->mesh().time().timeName(),
84                        this->mesh(),
85                        IOobject::NO_READ,
86                        IOobject::AUTO_WRITE
87                    ),
88                    tNAggdefault()
89                )
90            );
```

As seen in the above piece of code, `C_agg_sec` is initialized with a value of 0, and no further actions are required during the setup of the simulation case. For `N_agg_sec`, instead of specifying boundary conditions for individual instances like `N_agg_sec0`, `N_agg_sec1`, `N_agg_sec2`, ..., one should only specify the default boundary condition using the group name `N_agg`.

Because the number of fields depends on the `secNum` which is specified by the user, it is not possible to initialize the fields in the class constructor. Thus, only declaration is done in the class constructor and the initialization is performed by a class function named `create_fields()` that only includes `createSectionalFields.H`.

`laminarSPBM.C`

```
522 // Creating sectional fields
523 template<class ReactionThermo>
524 void Foam::combustionModels::laminarSPBM<ReactionThermo>::create_fields()
525 {
526     #include "createSectionalFields.H"
527 }
```

The declaration in the class constructor is shown in the following piece of code:

laminarSPBM.C

```
...
C_agg_sec(n_secs_.value()),
secNum(n_secs_.value()),
...
// Fields for tracked variables
N_agg_sec(n_secs_.value()),
N_pri_sec(n_secs_.value()),
C_tot_sec(n_secs_.value()),
...
// Fields for soot morphology
n_p_sec(n_secs_.value()),
m_agg_sec(n_secs_.value()),
d_p_sec(n_secs_.value()),
d_m_sec(n_secs_.value()),
d_g_sec(n_secs_.value()),
A_tot_sec(n_secs_.value()),
...
// Fields for source terms
I_grow_C_tot_sec(n_secs_.value()),
I_ox_C_tot_sec(n_secs_.value()),
I_coag_N_agg_sec(n_secs_.value()),
I_coag_N_pri_sec(n_secs_.value()),
...
// Fields for overall source terms
S_N_agg_sec(n_secs_.value()),
S_N_pri_sec(n_secs_.value()),
...
```

3.4.1 Sections definition function

For defining particle size sections, `PtrList<volScalarField>` data named `C_agg_sec` has been declared. The function blow uses Eq. (2.12) to calculate its value just once:

laminarSPBM.C

```
529 // Building sections
530 template<class ReactionThermo>
531 void Foam::combustionModels::laminarSPBM<ReactionThermo>::build_C_agg_sec()
532 {
533     forAll(secNum_, sec)
534     {
535         C_agg_sec_[sec] = C_min_ / Av_ * pow(spacing_, sec);
536     }
537 }
```

Additionally, some variables should be declared in the header file to be used in the section definition and looping over all sections.

laminarSPBM.H

```
104     // Number of sections
105     Foam::dimensionedScalar n_secs_;
106     // Spacing factor of sections
107     Foam::dimensionedScalar spacing_;
108     // Carbon per agglomerate in mole/#
109     PtrList<volScalarField> C_agg_sec_;
110     // list for forAll() loops
111     List<label> secNum_;
```

3.4.2 Tracked field functions

Four functions were implemented in `laminarSoot.H` to return the tracked fields named `N_agg()`, `N_pri()`, `C_tot()`, and `H_tot()`. Since they are no longer needed, they have been removed. Instead,

the `updateSootVariables()` function will be implemented and added to the `correct()` function to calculate the overall fields at each time step. The definition of the `updateSootVariables()` function will be provided in Section 3.17.

3.5 Gas properties

Functions responsible for calculating gas properties (`W(index)`, `C(index)`, and `lambda_gas()`) are independent of the size of the soot particles and should remain unchanged. The only identified issue pertains to a bug in the `lambda_gas()` function due to unit inconsistency, which is resolved by dividing `W()` by 1000 as shown in the below code.

laminarSPBM.H

```
270         virtual tmp<volScalarField> lambda_gas() const
271         {
272             return tmp<volScalarField>
273             (
274                 new volScalarField
275                 (
276                     max
277                     (
278                         this->thermo().mu() / this->thermo().rho() * pow(pi_ * this->thermo().W()
    /1000.0 / (2.0* kB_ * Av_ * this->thermo().T()), 0.5),
279                         dimensionedScalar(dimensionSet(0,1,0,0,0,0,0), SMALL)
280                     )
281                 )
282             );
283         };
```

3.6 PAH Characteristics

Similar to the gas functions, the functions that determine PAH properties (`m_PAH(id)` and `d_PAH(id)`) do not need modification, as they solely depend on PAH parameters.

3.7 Modifying `R(Y)` and `Qdot()` functions

Both `R(Y)` and `Qdot()` are default functions of the `laminar` combustion model that calculate the consumption/production rate of species and the rate of energy change due to the consumption/production of gas species through chemical reactions, respectively. However, the soot formation process involves the adsorption or release of certain chemical species into the gas phase. Therefore, in `laminarSoot`, some modifications have been made to incorporate the impact of adding/removing species by introducing a new variable, `SR_`, which represents the gas scrubbing rate. Fortunately, since all soot formation processes are identical in both MPBM and SPBM, the `SR_` values for all size sections are combined during the soot formation simulation. The resulting `SR_` can be used in `R(Y)` and `Qdot()` functions without any modification.

3.8 Modifying `correct()` function

Two additional functions should be included in the `correct()` function of `laminarSoot`. The first one, `updateSourceTerms()`, calculates the final source terms necessary for solving the transport equations (refer to Section 3.15 for implementation). The other one, `updateSootVariables()`, is called at the end of each time step to compute the overall values of various soot parameters (implementation in Section 3.17).

laminarSPBM.C

```
263         resetSR();
264         updateMorphology();
265         updateInception();
266         updateGrowth();
267         updateOxidation();
268         updateCoagulation();
269         updateSourceTerms();
270         updateSoot();
271         updateSootVariables();
```

Both of these two functions should be declared in the header file.

laminarSPBM.H

```
236         virtual void resetSR();
237         virtual void updateMorphology();
238         virtual void updateInception();
239         virtual void updateGrowth();
240         virtual void updateOxidation();
241         virtual void updateCoagulation();
242         virtual void updateSourceTerms();
243         virtual void updateSoot();
244         virtual void updateSootVariables();
```

3.9 Resetting scrubbing rates by resetSR()

The resetSR() function in laminarSoot sets the scrubbing rate of all species to zero at the beginning of each time step, and it can function in the same manner for laminarSPBM.

3.10 updateMorphology() function

Two functions implemented in the laminarSoot.H file to calculate the volume and the mass of the agglomerates (V_agg() and m_agg()) should be removed as the calculation of mass and volume should be done for each section if needed. In laminarSPBM, updating morphology of the soot particles is performed by updateMorphology functions to determine morphological variables n_{p}^{i}, d_{p}^{i}, d_{m}^{i}, and d_{g}^{i} for all section sizes. The calculations are based on the equations provided in Section 2.1.

laminarSPBM.C

```
641 // Updating morphology - Sectional
642 template<class ReactionThermo>
643 void Foam::combustionModels::laminarSPBM<ReactionThermo>::updateMorphology()
644 {
645     forAll(secNum_, i)
646     {
647         //  Mobility diameter
648         d_m_sec_[i] = max(d_p_sec_[i], d_p_sec_[i] * pow(n_p_sec_[i], 0.45));
649 
650         // Gyration Diameter
651         volScalarField n_p_lowerlimit (n_p_sec_[i]*0.0+1.5);
652         d_g_sec_[i] = (n_p_sec_[i] <= n_p_lowerlimit) * (d_m_sec_[i] / 1.29) + \
653         (n_p_sec_[i] > n_p_lowerlimit) * (d_m_sec_[i] / (pow(n_p_sec_[i], -0.2)+0.4));
654 
655         // Surface area of each primary particle
656         A_tot_sec_[i] = N_pri_sec_[i] * Av_ * pi_ * d_p_sec_[i] * d_p_sec_[i];
657 
658         // Primary particle diameter
659         d_p_sec_[i] = pow ((6.0 / pi_) * (C_agg_sec_[i] * W_carbon_) / (rho_soot_ * n_p_sec_[i]),
     1.0/3.0);
660 
661         // number of primary particles
```

```
662         n_p_sec_[i] = min(max(N_pri_sec_[i] / N_agg_sec_[i], 1.0), C_agg_sec_[i] / C_agg_sec_[0]);
663
664         // mass of each agglomerate
665         m_agg_sec_[i] = rho_soot_ * n_p_sec_[i] * (pi_ / 6.0) * pow(d_p_sec_[i], 3.0);
666     }
667
668 }
```

3.11 Updating Inception Source Term

3.11.1 `updateInception()` fuction

Inception was modeled using the reactive dimerization method in `laminarSoot`. This implies that the rate of production of chemically-bonded dimers is calculated, and various source terms can be determined based on the rate of production and other corresponding parameters. However, in SPBM, the inception source term only affects the first section, and its contribution to N_{agg} and N_{pri} is determined by Eq. (2.14). Therefore, the `updateInception()` function should be slightly modified by removing `S_inc_N_` and `S_inc_H_tot_`. Please note that although `laminarSPBM` does not track hydrogen, the removal of hydrogen atoms is considered in the scrubbing rate variable, `SR_`.

laminarSPBM.C

```
670 // Updating inception source terms
671 template<class ReactionThermo>
672 void Foam::combustionModels::laminarSPBM<ReactionThermo>::updateInception()
673 {
674     I_inc_C_tot_ *= 0.0;
675     if (inception_enabled_){
676         volScalarField rho = this->thermo().rho();
677         forAll(dimer_names_, i)
678         {
679             // PAH Index and Id
680             label id1 = dimer_PAH_1_id_[i];
681             label id2 = dimer_PAH_2_id_[i];
682             volScalarField dimerROPField(dimerROP(id1, id2));
683             // C_tot Source Term
684             I_inc_C_tot_ += dimer_n_C_[i] * dimerROPField / rho;
685
686             if (scrubbing_enabled_){
687                 // PAHs
688                 // species id
689                 label spid1 = speciesIds_[PAH_names_[id1]];
690                 label spid2 = speciesIds_[PAH_names_[id2]];
691                 // species index
692                 label spindex1 = speciesIndicies_[PAH_names_[id1]];
693                 label spindex2 = speciesIndicies_[PAH_names_[id2]];
694                 SR_[spid1] -= dimerROPField * W(spindex1);
695                 SR_[spid2] -= dimerROPField * W(spindex2);
696
697                 // H2
698                 label H2_id = speciesIds_["H2"];
699                 label H2_index = speciesIndicies_["H2"];
700                 SR_[H2_id] += dimerROPField * W(H2_index);
701             }
702         }
703     }
704 }
```

Because dimerization is independent of particle size and only depends on the properties of PAHs and the reaction rates, all functions related to the inception process (`dimerROP(id1, id2)`, `k_FWD(id1, id2)`, `k_REV(id1, id2)`, and `k_REAC()`) should remain unchanged.

3.12 Updating soot growth rates

For calculating the growth source term for each section, first `HACAGrowthRate(sec)` calculates the HACA rates, and then the contribution of the HACA is determined based on Eq. (A.16). Next, for each section, `updateGrowth()` loops over all PAHs and computes the adsorption rate for each PAH (using `PAHAdsorptionRate(sec, id)` function) and adds it to the growth source terms. After each loop, if scrubbing is enabled, `SR_` is updated to consider the consumption or production of the gas species. Please note that Both HACA and PAH adsorption mechanisms depend on the particle size; thus, they should be updated for each section. Again, the removal of hydrogen is taken into account while updating the scrubbing rate.

laminarSPBM.C

```
708 // Updating growth source terms
709 template<class ReactionThermo>
710 void Foam::combustionModels::laminarSPBM<ReactionThermo>::updateGrowth()
711 {
712     forAll(secNum_, sec)
713     {
714         I_grow_C_tot_sec_[sec] *= 0.0;
715 
716         if (HACA_growth_enabled_)
717         {
718         volScalarField rho = this->thermo().rho();
719         volScalarField HACAGrowthRateField(HACAGrowthRate(sec));
720         I_grow_C_tot_sec_[sec] += 2 * HACAGrowthRateField / rho;
721 
722             if (scrubbing_enabled_){
723             // C2H2
724             label C2H2_id = speciesIds_["C2H2"];
725             label C2H2_index = speciesIndicies_["C2H2"];
726             SR_[C2H2_id] -= HACAGrowthRateField * W(C2H2_index);
727 
728             // H
729             label H_id = speciesIds_["H"];
730             label H_index = speciesIndicies_["H"];
731             SR_[H_id] += HACAGrowthRateField * W(H_index) * (1.75 / 2.00);
732         }
733         }
734         if (PAH_growth_enabled_)
735         {
736         volScalarField rho = this->thermo().rho();
737         forAll(PAH_names_, id)
738         {
739             volScalarField PAHAdsorptionRateField(PAHAdsorptionRate(sec, id));
740             I_grow_C_tot_sec_[sec] += PAH_n_C_[id] * PAHAdsorptionRateField / rho;
741 
742             if (scrubbing_enabled_){
743                 // PAH
744                 // species id
745                 label spid = speciesIds_[PAH_names_[id]];
746                 // species index
747                 label spindex = speciesIndicies_[PAH_names_[id]];
748                 SR_[spid] -= PAHAdsorptionRateField * W(spindex);
749 
750                 // H
751                 label H_id = speciesIds_["H"];
752                 label H_index = speciesIndicies_["H"];
753                 SR_[H_id] += PAHAdsorptionRateField * W(H_index) * 2;
754             }
755         }
756         }
757     }
758 }
```

3.12.1 `HACAGrowthRate(sec)` function

Below code is the implemented `HACAGrowthRate(sec)` that calculates the rate of acetylene addition through the HACA mechanism for each section. For more information refer to Section A.2.

laminarSPBM.H

```
511         virtual tmp<volScalarField> HACAGrowthRate(label sec)
512         {
513             label C2H2_i = speciesIndicies_["C2H2"];
514             return tmp<volScalarField>
515             (
516                 new volScalarField
517                 (
518                     max
519                     (
520                         alpha(sec) * k_4_HACA() * C(C2H2_i) * C_soot_0(sec),
521                         dimensionedScalar(dimensionSet(0,-3,-1,0,1,0,0), scalar(0.0))
522                     )
523                 )
524             );
525         }
```

As can be seen in the above piece of code, `alpha(sec)` and `C_soot_0(sec)` functions are section-specific. Thus, some minor modifications are needed to perform for these two functions.
For the `C_soot_0(sec)` function:

laminarSPBM.H

```
474             return tmp<volScalarField>
475             (
476                 new volScalarField
477                 (
478                     "C_soot_0",
479                     (A_tot_sec_[sec] / Av_ * chi_soot_0) * rho
480                 )
481             );
482         }
```

For the `alpha(sec)` function:

laminarSPBM.H

```
484         virtual tmp<volScalarField> alpha(label sec)
485         {
486             dimensionedScalar oneKelvin(dimTemperature, scalar(1.0));
487             dimensionedScalar alpha_min(dimless, scalar(0.0));
488             dimensionedScalar alpha_max(dimless, scalar(1.0));
489             const volScalarField& T = this->thermo().T();
490             return tmp<volScalarField>
491             (
492                 new volScalarField
493                 (
494                     min
495                     (
496                         max
497                         (
498                             tanh
499                             (
500                                 (12.56 - 0.00563 * T / oneKelvin) / log10 ( rho_soot_ * pi_ / 6.0 *
    pow(d_p_sec_[sec], 3.0) * Av_ / W_carbon_ ) -
501                                 1.38 + 0.00068 * T / oneKelvin
502                             ),
503                             alpha_min
504                         ),
505                         alpha_max
506                     )
507                 )
```

```
            );
        }
```

3.12.2 `PAHAdsorptionRate(sec, id)` function

Below code is the implemented `PAHAdsorptionRate(sec, id)` that calculates the rate of PAH adsorption for a given PAH and particle section. For more information refer to Section A.3.

laminarSPBM.H

```
        virtual tmp<volScalarField> PAHAdsorptionRate(label sec, 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_sec_[sec] + 1.0/m_PAH(id)), 0.5) * pow(
    d_g_sec_[sec] + d_PAH(id), 2.0)
            );
            // beta cont coag PAH-soot
            volScalarField C_s_soot_d_m
            (
                1.0 + (2.0 * lambda_gas() / d_m_sec_[sec] ) * (1.21 + 0.4*exp(-0.78 * d_m_sec_[sec] /
    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_sec_[sec] + C_s_soot_d_PAH / d_PAH(id
    )) * (d_g_sec_[sec] + 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_agg_sec_[sec] * W_carbon_ * Av_);//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))
```

```
615             );
616
617             // Chemical adsorption rate
618             dimensionedScalar Ea_k_REAC_soot_PAH(dimensionSet(1,2,-2,0,-1,0,0), scalar(96232.0));
619             dimensionedScalar A_k_REAC_soot_PAH(dimensionSet(0,3,-1,0,-1,0,0), scalar(2.0e10));
620             volScalarField k_REAC_soot_PAH
621             (
622                 A_k_REAC_soot_PAH * exp(-Ea_k_REAC_soot_PAH / (Ru_ * T))
623             );
624             return tmp<volScalarField>
625             (
626                 new volScalarField
627                 (
628                     max
629                     (
630                         k_REAC_soot_PAH * k_FWD_soot_PAH * C(index) * (N_agg_sec_[sec] * rho) / (
    k_REAC_soot_PAH + k_REV_soot_PAH),
631                         dimensionedScalar(dimensionSet(0,-3,-1,0,1,0,0), scalar(0.0))
632                     )
633                 )
634             );
635         }
```

3.13 Updating oxidation rates

3.13.1 `updateOxidation()` function

The following piece of code shows `updateOxidation()` function that determines the rate of oxidation of soot particles in each section based on `HACAO2OxidationRate(sec)` and `HACAOHOxidationRate(sec)` functions. Finally, if scrubbing is enabled, the addition and removal of the involved chemical species are included in `SR_`. The detailed formulation of the oxidation process is presented in Section A.4.

laminarSPBM.C

```
761 template<class ReactionThermo>
762 void Foam::combustionModels::laminarSPBM<ReactionThermo>::updateOxidation()
763 {
764     forAll(secNum_, sec)
765     {
766         I_ox_C_tot_sec_[sec] *= 0.0;
767         if (HACA_oxidation_enabled_){
768         volScalarField rho(this->thermo().rho());
769         volScalarField HACAO2OxidationRateField(HACAO2OxidationRate(sec));
770         volScalarField HACAOHOxidationRateField(HACAOHOxidationRate(sec));
771         I_ox_C_tot_sec_[sec] += -1 * (HACAO2OxidationRateField + HACAOHOxidationRateField) / rho;
772
773         if (scrubbing_enabled_){
774             // O2
775             label O2_id = speciesIds_["O2"];
776             label O2_index = speciesIndicies_["O2"];
777             SR_[O2_id] -= 0.5 * HACAO2OxidationRateField * W(O2_index);
778
779             // CO2
780             label CO_id = speciesIds_["CO"];
781             label CO_index = speciesIndicies_["CO"];
782             SR_[CO_id] += (HACAO2OxidationRateField + HACAOHOxidationRateField) * W(CO_index);
783
784             // OH
785             label OH_id = speciesIds_["OH"];
786             label OH_index = speciesIndicies_["OH"];
787             SR_[OH_id] -= HACAOHOxidationRateField * W(OH_index);
788         }
789         }
790     }
791 }
```

3.13.2 `HACAO2OxidationRate(sec)` and `HACAOHOxidationRate(sec)` function

HACA oxidation rates by O_2 and OH are determined by the following code which shows implementation of `HACAO2OxidationRate(sec)` and `HACAOHOxidationRate(sec)` functions.

laminarSPBM.H

```
527         virtual tmp<volScalarField> HACAO2OxidationRate(label sec)
528         {
529             label O2_i = speciesIndicies_["O2"];
530             volScalarField rho (this->thermo().rho());
531             return tmp<volScalarField>
532             (
533                 new volScalarField
534                 (
535                     max
536                     (
537                         2 * alpha(sec) * k_5_HACA() * C(O2_i) * C_soot_0(sec),
538                         dimensionedScalar(dimensionSet(0,-3,-1,0,1,0,0), scalar(0.0))
539                     )
540                 )
541             );
542         }
543 
544         virtual tmp<volScalarField> HACAOHOxidationRate(label sec)
545         {
546             label OH_i = speciesIndicies_["OH"];
547             volScalarField rho (this->thermo().rho());
548             return tmp<volScalarField>
549             (
550                 new volScalarField
551                 (
552                     max
553                     (
554                         k_6_HACA() * C(OH_i) * N_agg_sec_[sec] * rho,  // Mo
555                         dimensionedScalar(dimensionSet(0,-3,-1,0,1,0,0), scalar(0.0))
556                     )
557                 )
558             );
559         }
```

3.14 Updating coagulation rate

As discussed in Section 2.4.4, coagulation alters the number of primary particles and agglomerates in each section. The source terms that apply the impact of coagulation in the transport equations are computed by `updateCoagulation()`.

laminarSPBM.C

```
794 template<class ReactionThermo>
795 void Foam::combustionModels::laminarSPBM<ReactionThermo>::updateCoagulation()
796 {
797 
798     volScalarField rho (this->thermo().rho());
799 
800     // Coagulation source term
801     if (coagulation_enabled_)
802     {
803     volScalarField C_agg_sec_curr (C_agg_sec_[0]*0.0);
804     volScalarField C_agg_sec_next (C_agg_sec_[0]*0.0);
805     volScalarField C_agg_sec_prev (C_agg_sec_[0]*0.0);
806     volScalarField eta_ijk (n_p_sec_[0]*0.0);
807     volScalarField eta_p_ijk (n_p_sec_[0]*0.0);
808 
809     double coag_prefactor = 0.0;
```

```
810
811     dimensionedScalar beta("beta", dimensionSet(0, 3, -1, 0, 0, 0, 0),  scalar(1e-15));
812
813         forAll(secNum_, sec)
814         {
815             // Resetting sum1 variables
816             volScalarField sum1_N_agg (sum1_inside_*0.0);
817             volScalarField sum1_N_pri (sum1_inside_*0.0);
818
819             // C_agg_sec of the current section
820             C_agg_sec_curr = C_agg_sec_[sec];
821
822             // The C_agg_sec for the first section is the same as C_agg_sec_curr
823             if(sec==0){
824                 C_agg_sec_prev = C_agg_sec_curr;
825             }
826             else
827             {
828                 C_agg_sec_prev = C_agg_sec_[sec-1];
829             }
830             // The C_agg_sec for the last section is the same as spacing_*C_agg_sec_curr
831             if(sec==(n_secs_.value()-1)){
832                 C_agg_sec_next = spacing_*C_agg_sec_curr;
833             }
834             else
835             {
836                 C_agg_sec_next = C_agg_sec_[sec+1];
837             }
838
839             // Addition of particles to section
840             for (int k = 0; k <= sec; k++)
841             {
842                 for (int j = k; j <= sec; j++)
843                 {
844                     volScalarField C_agg_sec_jk (C_agg_sec_[j] + C_agg_sec_[k]);
845                     if (C_agg_sec_prev <= C_agg_sec_jk && C_agg_sec_jk <= C_agg_sec_next)
846                     {
847                         // Calculating eth_ijk
848                         if (C_agg_sec_curr < C_agg_sec_jk && C_agg_sec_jk < C_agg_sec_next)
849                         {
850                             eta_ijk = (C_agg_sec_next-C_agg_sec_jk)/(C_agg_sec_next-C_agg_sec_curr);
851                         }
852                         else if (C_agg_sec_prev < C_agg_sec_jk && C_agg_sec_jk < C_agg_sec_curr)
853                         {
854                             eta_ijk = (C_agg_sec_prev-C_agg_sec_jk)/(C_agg_sec_prev-C_agg_sec_curr);
855                         }
856
857                         // Calculation eta_p_ijk (NO MERGING)
858                         eta_p_ijk = C_agg_sec_curr / C_agg_sec_jk * (n_p_sec_[j] + n_p_sec_[k]);
859
860                         // Corresponds to 1-delta(j,k)/2
861                         if (j==k)
862                         {
863                             coag_prefactor = 0.5;
864                         }
865                         else
866                         {
867                             coag_prefactor = 1.0;
868                         }
869
870                         sum1_inside_ = coag_prefactor * eta_ijk * beta * N_agg_sec_[j] * N_agg_sec_[k
    ];
871                         sum1_N_agg += sum1_inside_;
872                         sum1_N_pri += sum1_inside_ * eta_p_ijk;
873                     }
874
875                 }
876             }
```

```
877
878             // Removal of particles to section
879             sum2_all_ *= 0.0;
880             for (int m = 0; m < n_secs_.value(); m++)
881             {
882                 sum2_all_ += beta * N_agg_sec_[m];
883             }
884
885             // Coagulation source terms
886             I_coag_N_agg_sec_[sec] = (sum1_N_agg - N_agg_sec_[sec] * sum2_all_) * Av_ * rho;
887             I_coag_N_pri_sec_[sec] = (sum1_N_pri - N_pri_sec_[sec] * sum2_all_) * Av_ * rho;
888         }
889
890     }
891
892 }
```

3.15 Updating source terms

The overall source terms for each section can be derived by combining the individual source terms computed for each of the mechanisms. The implementation of the class function responsible for calculating the final source terms (`S_N_agg_sec` and `S_N_pri_sec`) is based on Eq. (2.16) and Eq. (2.17) for surface growth, and Eq. (2.19) and Eq. (2.20) for the oxidation process. The implementation of the class function `updateSourceTerms()` is presented below.

laminarSPBM.C

```
908 template<class ReactionThermo>
909 void Foam::combustionModels::laminarSPBM<ReactionThermo>::updateSourceTerms()
910 {
911     forAll(secNum_, sec)
912     {
913         S_N_agg_sec_[sec] *= 0.0;
914         S_N_pri_sec_[sec] *= 0.0;
915
916         // First section
917         if(sec==0)
918         {
919             // Inception
920             S_N_agg_sec_[sec] += I_inc_C_tot_ / C_agg_sec_[0] / Av_;
921             S_N_pri_sec_[sec] += I_inc_C_tot_ / C_agg_sec_[0] / Av_;
922
923             // PAH adsorption & surface growth
924             S_N_agg_sec_[sec] += - I_grow_C_tot_sec_[0] / (C_agg_sec_[1] - C_agg_sec_[0]) / Av_;
925             S_N_pri_sec_[sec] += - I_grow_C_tot_sec_[0] / (C_agg_sec_[1] - C_agg_sec_[0]) / Av_;
926
927             // Oxidation
928             S_N_agg_sec_[sec] += (I_ox_C_tot_sec_[1] / (C_agg_sec_[1] - C_agg_sec_[0]) \
929             - I_ox_C_tot_sec_[0] / C_agg_sec_[0]) / Av_;
930             S_N_pri_sec_[sec] += (I_ox_C_tot_sec_[1] / (C_agg_sec_[1] - C_agg_sec_[0]) * n_p_sec_[1] \
931             - I_ox_C_tot_sec_[0] / C_agg_sec_[0]) / Av_;
932
933             // Coagulation
934             S_N_agg_sec_[sec] += I_coag_N_agg_sec_[0];
935             S_N_pri_sec_[sec] += I_coag_N_pri_sec_[0];
936         }
937
938         // Middle sections
939         else if (sec > 0 && sec < (this->n_secs_.value() - 1))
940         {
941             // PAH adsorption & surface growth
942             S_N_agg_sec_[sec] += (I_grow_C_tot_sec_[sec-1] / (C_agg_sec_[sec] - C_agg_sec_[sec-1])\
943             - I_grow_C_tot_sec_[sec] / (C_agg_sec_[sec+1] - C_agg_sec_[sec])) / Av_;
944             S_N_pri_sec_[sec] += (I_grow_C_tot_sec_[sec-1] / (C_agg_sec_[sec] - C_agg_sec_[sec-1]) *
    n_p_sec_[sec-1] \
```

```
945             - I_grow_C_tot_sec_[sec] / (C_agg_sec_[sec+1] - C_agg_sec_[sec]) * n_p_sec_[sec]) / Av_;
946
947             // Oxidation
948             S_N_agg_sec_[sec] += (I_ox_C_tot_sec_[sec+1] / (C_agg_sec_[sec+1] - C_agg_sec_[sec]) \
949             - I_ox_C_tot_sec_[sec] / (C_agg_sec_[sec] - C_agg_sec_[sec-1])) / Av_;
950             S_N_pri_sec_[sec] += (I_ox_C_tot_sec_[sec+1] / (C_agg_sec_[sec+1] - C_agg_sec_[sec]) *
    n_p_sec_[sec+1]\
951             - I_ox_C_tot_sec_[sec] / (C_agg_sec_[sec] - C_agg_sec_[sec-1]) * n_p_sec_[sec]) / Av_;
952
953             // Coagulation
954             S_N_agg_sec_[sec] += I_coag_N_agg_sec_[sec];
955             S_N_pri_sec_[sec] += I_coag_N_pri_sec_[sec];
956         }
957
958         // Last section
959         else if (sec == this->n_secs_.value() - 1)
960         {
961             // PAH adsorption & surface growth
962             S_N_agg_sec_[sec] += I_grow_C_tot_sec_[sec-1] / (C_agg_sec_[sec] - C_agg_sec_[sec-1]) /
    Av_;
963             S_N_pri_sec_[sec] += I_grow_C_tot_sec_[sec-1] / (C_agg_sec_[sec] - C_agg_sec_[sec-1]) *
    n_p_sec_[sec-1] / Av_;
964
965             // Oxidation
966             S_N_agg_sec_[sec] += - I_ox_C_tot_sec_[sec] / (C_agg_sec_[sec] - C_agg_sec_[sec-1]) / Av_;
967             S_N_pri_sec_[sec] += - I_ox_C_tot_sec_[sec] / (C_agg_sec_[sec] - C_agg_sec_[sec-1]) *
    n_p_sec_[sec] / Av_;
968
969             // Coagulation
970             S_N_agg_sec_[sec] += I_coag_N_agg_sec_[sec];
971             S_N_pri_sec_[sec] += I_coag_N_pri_sec_[sec];
972         }
973
974     }
975 }
```

3.16 Solving the transport equations

In the sectional population balance model, the values of N_{agg} and N_{pri} in each section are obtained by solving transport equations employing the corresponding source terms. However, since the size of each section is predefined, the value of C_{tot} for each section can be easily determined by an algebraic equation. The function `updateSoot()` solves the three desired soot variables for each section by iterating over all sections.

laminarSPBM.C

```
978 template<class ReactionThermo>
979 void Foam::combustionModels::laminarSPBM<ReactionThermo>::updateSoot()
980 {
981     const surfaceScalarField& phi = this->phi();
982     volScalarField rho (this->thermo().rho());
983     fv::options& fvOptions(fv::options::New(this->mesh_));
984
985     forAll(secNum_, sec)
986     {
987         const volScalarField D(diffusionCoeff(sec));
988         // N_agg_ Equation
989         {
990         Info<< "N_agg Equation for section " << sec << endl;
991         volScalarField& N_agg = N_agg_sec_[sec];
992         fvScalarMatrix N_aggEqn
993         (
994             fvm::ddt(rho, N_agg)
995             + fvm::div(phi, N_agg)
996             - fvm::laplacian(D*rho, N_agg)
```

```
997             ==
998             rho * S_N_agg_sec_[sec]
999         );
1000
1001        N_aggEqn.relax();
1002        fvOptions.constrain(N_aggEqn);
1003        N_aggEqn.solve(this->mesh().solver("N_agg"));
1004        fvOptions.correct(N_agg);
1005        }
1006
1007        // N_pri_ Equation
1008        {
1009        Info<< "N_pri Equation for section " << sec << endl;
1010        volScalarField& N_pri = N_pri_sec_[sec];
1011        fvScalarMatrix N_priEqn
1012        (
1013            fvm::ddt(rho, N_pri)
1014            + fvm::div(phi, N_pri)
1015            - fvm::laplacian(D*rho, N_pri)
1016            ==
1017            rho * S_N_pri_sec_[sec]
1018        );
1019
1020        N_priEqn.relax();
1021        fvOptions.constrain(N_priEqn);
1022        N_priEqn.solve(this->mesh().solver("N_pri"));
1023        fvOptions.correct(N_pri);
1024        }
1025
1026        // C_tot Equation
1027        {
1028            Info<< "C_tot Equation for section " << sec << endl;
1029            C_tot_sec_[sec] = N_agg_sec_[sec] * Av_ * C_agg_sec_[sec];
1030        }
1031    }
1032
1033 }
```

3.17 Overall values calculation

After calculating all soot variables for each section, overall values can be computed by summation over all sections for extensive variables, and by number based averaging for intensive variables. In the `laminarSPBM` library, the `updateSootVariables()` function is implemented to determine the overall value of N_{agg}, N_{pri}, C_{tot}, and d_{m}.

laminarSPBM.C

```
1036 template<class ReactionThermo>
1037 void Foam::combustionModels::laminarSPBM<ReactionThermo>::updateSootVariables()
1038 {
1039     N_agg_ *= 0.0;
1040     N_pri_ *= 0.0;
1041     C_tot_ *= 0.0;
1042     volScalarField d_m_sum (d_m_*N_agg_*0.0);
1043
1044     forAll(secNum_, sec)
1045     {
1046         N_agg_ += N_agg_sec_[sec];
1047         N_pri_ += N_pri_sec_[sec];
1048         C_tot_ += C_tot_sec_[sec];
1049         d_m_sum += N_agg_sec_[sec] * d_m_sec_[sec];
1050     }
1051
1052     d_m_ = d_m_sum / N_agg_;
1053 }
```

Evidently, All required fields should be declared in the header file and initialized in the class constructor.

laminarSPBM.H

```
144     volScalarField N_agg_;
145     volScalarField N_pri_;
146     volScalarField C_tot_;
147     volScalarField d_m_;
```

laminarSPBM.C

```
225     N_agg_
226     (
227         IOobject
228         (
229             "N_agg",
230             this->mesh().time().timeName(),
231             this->mesh(),
232             IOobject::NO_READ,
233             IOobject::AUTO_WRITE
234         ),
235         this->mesh(), dimensionedScalar("N_agg", dimensionSet(-1,0,0,0,1,0,0),0.0)
236     ),
```

Chapter 4

Tutorial

4.1 Physics and Geometry

As the main focus of the tutorial is on flow chemistry and the soot formation process, a simple case study of a zero-dimensional constant volume reactor is set up to test the `laminarSPBM`. Using such a simple case is sufficient for the evaluation stage while maintaining an affordable computational cost. The computational domain consists of just 36 cells uniformly distributed, with no inlet or outlet considered for the reactor, as portrayed in Fig. 4.1. The reactor, being a closed box, has an initial velocity of zero in both directions, leading to zero velocity during the simulation time. This means all fields will be uniform across the domain.

The gas is initially at a pressure of $P = 1.0 \times 10^5$ Pa and a temperature of $T = 1800$ K. The initial composition of the gas is set by specifying the mass fraction of CH_4 and N_2, and the mass fraction of the rest of the species is set to zero. Table 4.1 summarizes the initial and boundary conditions for all fields.

In the following section, a detailed description will be provided for setting up the simulation case. The reader is encouraged to refer to the tutorial case accompanying this report for more instructions about the simulation case.

4.2 Time Directory

4.2.1 Gas Phase

The initial composition of the gas is defined by setting up the mass fraction of methane $Y_{CH_4} = 0.3$ and nitrogen $Y_{N_2} = 0.7$. The mass fraction of all other species is set to be zero in the `Ydefault` file. The boundary condition for `walls` patch is `zeroGradient` for all fields. The content of the species

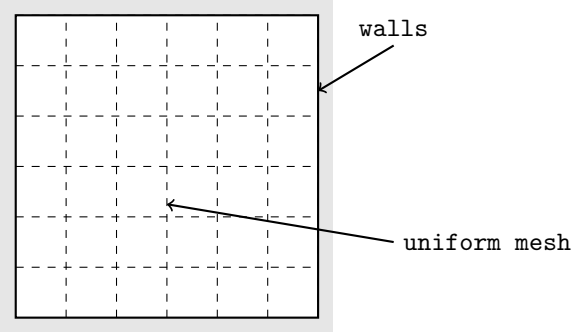

Figure 4.1: Computational domain with 6×6 uniform grid.

Field	Initial value	Boundary condition
U	(0 0 0)	zeroGradient
P	1.0e5	zeroGradient
T	1800	zeroGradient
N2	0.7	zeroGradient
CH4	0.3	zeroGradient
Ydefault	0	zeroGradient

Table 4.1: Initial and boundary conditions

files located in 0 directory is provided below.

0/Ch4

```
/*--------------------------------*- 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 0.3;

boundaryField
{
    walls
    {
        type            zeroGradient;
    }
    frontAndBack
    {
        type            empty;
    }
}
// ************************************************************************* //
```

For the N2 file, similar content should be included.

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 0.7;

boundaryField
{
    walls
    {
        type            zeroGradient;
    }
    frontAndBack
    {
        type            empty;
    }
}


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

The content of the `Ydefault` file indicates that initially, the gas is the mixture of only methane and nitrogen.

0/Ydefault

```
/*--------------------------------*- 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      Ydefault;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

internalField   uniform 0.0;

boundaryField
{
    frontAndBack
    {
        type            empty;
    }
    walls
    {
        type            zeroGradient;
    }
}
// ************************************************************************* //
```

Boundary and initial conditions for pressure and temperature are similar to the mass fraction files but consistent with the values specified in Table 4.1.

4.2.2 Solid Phase

Note that `laminarSPBM` requires `N_agg_sec` and `N_pri_sec` for each section, but as all of `N_agg_sec` and `N_pri_sec` are grouped together under the names `N_agg` and `N_pri`, one needs to set up only two files, `N_agg` and `N_pri`, in the `0` directory. To avoid division by zero, it is assumed that there is one agglomerate consisting of just one primary particle in each section. This infinitesimal initial value, which will not have any impact on the results, has the unit of moles per kilogram of the gas mixture and is calculated as follows.

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

The boundary conditions of both `N_agg` and `N_pri` files are similar to the mass fractions as can be seen in the following.

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
{
    walls
    {
        type            zeroGradient;
    }
    frontAndBack
    {
        type            empty;
    }
}
// ************************************************************************* //
```

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
{
    walls
    {
        type            zeroGradient;
    }
    frontAndBack
    {
        type            empty;
    }
}
// ************************************************************************* //
```

4.3 Constant Directory

In OpenFOAM, the combustion model is configured in the `combustionProperties` file, which is located in the `constant` directory. Here, the `laminarSPBM` model is specified to utilize the sectional soot model.

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  laminarSPBM;

active  true;

laminarCoeffs
{
}
// ************************************************************************* //
```

Soot model settings are configured in the `sootProperties` dictionary. This setup involves specifying PAHs treated as soot precursors, variables required for creating sections and switches for activating/deactivating various mechanisms. The settings used in this tutorial are outlined below.

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);

numberOfSections    40;
spacingFactor       1.5;

scrubbing_enabled   true;
PAH_growth_enabled  true;
HACA_growth_enabled true;
inception_enabled   true;
HACA_oxidation_enabled true;
coagulation_enabled true;

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

In this tutorial, the Appel-Beerhens-Fenske (ABF) mechanism [9] is employed to solve the gas chemistry and obtain thermophysical properties. Consequently, both the thermophysical and chemistry files, named `thermo.ABF-mod` and `reactions.ABF` respectively, formatted in Chemkin, are placed in the `constant` directory. Subsequently, the names of these two files are specified in the `thermophysicalProperties` to be utilized by `foamChemistryReader`.

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";
```

```
33 foamChemistryThermoFile "<constant>/thermo.ABF-mod";
34
35 // ************************************************************************* //
```

4.4 System Directory

The simulation is planned to run for $t = 0.2s$ with a time step of $\mathrm{d}t = 1 \times 10^{-4}s$, as indicated in the `controDict` dictionary below. Additionally, `liblaminarSPBM.so` is included to enable the solver to link to the new library. For postprocessing purposes, the `probes` function object is also included.

system/controlDict

```
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      controlDict;
15 }
16 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18 application     reactingFoam;
19 startFrom       latestTime;
20 startTime       0;
21 stopAt          endTime;
22 endTime         0.2;
23 deltaT          1e-4;
24 writeControl    timeStep;
25 writeInterval   100;
26 purgeWrite      2;
27 writeFormat     ascii;
28 writePrecision  7;
29 writeCompression off;
30 timeFormat      general;
31 timePrecision   6;
32 runTimeModifiable true;
33 adjustTimeStep  no;
34 maxCo           0.1;
35 functions{
36 #include "probes"
37 }
38 libs ("liblaminarSPBM.so");
39 // ************************************************************************* //
```

In the `fvSchemes` dictionary, the default value for `divSchemes` is set to `Gauss limitedLinear 1`. Consequently, there is no need to specify `divSchemes` for `N_agg_sec` and `N_pri_sec` of each section separately.

system/fvSchemes

```
1
2  /*--------------------------------*- C++ -*----------------------------------*\
3  | =========                 |                                                 |
4  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
5  |  \\    /   O peration     | Version:  v2006                                 |
6  |   \\  /    A nd           | Website:  www.openfoam.com                      |
7  |    \\/     M anipulation  |                                                 |
```

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

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
}

divSchemes
{
    default         Gauss limitedLinear 1;

    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(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear orthogonal;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         orthogonal;
}
```

In the `fvSolution` dictionary, solution control entries for both group names `N_agg` and `N_pri` should be added.

system/fvSolution

```

/*--------------------------------*- 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      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    "rho.*"
    {
        solver          diagonal;
    }

    p
    {
        solver           PCG;
        preconditioner   DIC;
        tolerance        1e-6;
        relTol           0.1;
    }

    pFinal
    {
        $p;
        tolerance        1e-6;
        relTol           0.0;
    }

    "(U|h|N_agg|N_pri)"
    {
        solver          PBiCGStab;
        preconditioner  DILU;
        tolerance       1e-6;
        relTol          0.1;
    }

    "(U|h|N_agg|N_pri)Final"
    {
        $U;
        relTol          0;
    }

    Yi
    {
        $hFinal;
    }
}

PIMPLE
{
    momentumPredictor no;
    nOuterCcorrectors  1;
    nCorrectors     2;
    nNonOrthogonalCorrectors 0;
}
// ************************************************************************* //
```

A `probes` sampling file is added to the `system` directory to collect simulation data over time at the center of the domain, with `probeLocations` set to `(0.05 0.05 0.005)`. The fields to be probed include the total number of agglomerates (`N_agg`), total number of primary particles (`N_pri`), and total carbon content (`C_tot`). Additionally, to enable the plotting of the particle size distribution, N_{agg} and N_{pri} of each section are collected during the simulation time. Since the number of sections is not fixed and can be modified by the user, a pre-processing Python code has been developed to read the number of sections, `numberOfSections`, from the `sootProperties` dictionary and make the necessary adjustments to the value of the `fields` entry. The following is the `probes` file if the number of sections is set to five.

`system/probes`

```
1
2  /*--------------------------------*- C++ -*----------------------------------*\
3  | =========                 |                                                 |
4  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
5  |  \\    /   O peration     | Version:  v2006                                 |
6  |   \\  /    A nd           | Website:  www.openfoam.com                      |
7  |    \\/     M anipulation  |                                                 |
8  \*---------------------------------------------------------------------------*/
9  FoamFile
10 {
11     version     2.0;
12     format      ascii;
13     class       dictionary;
14     location    "system";
15     object      probesSampling;
16 }
17 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
18
19 probes
20 {
21     type            probes;
22     libs            (sampling);
23     name            probes;
24     writeControl    outputTime;
25     writeInterval   1;
26     interpolationScheme cellPoint;
27     sampleOnExecute yes;
28
29     fields         (N_agg N_pri C_tot N_agg_sec0 N_agg_sec1 N_agg_sec2 N_agg_sec3 N_agg_sec4
        N_pri_sec0 N_pri_sec1 N_pri_sec2 N_pri_sec3 N_pri_sec4);
30
31     probeLocations
32     (
33         (0.05 0.05 0.005)
34     );
35 }
```

4.5 Running the case

The computational grid is generated using the `blockMesh` utility, and the simulation is carried out in serial mode. An `Allrun` script runs all necessary commands, including those needed for mesh generation, preprocessing to adjust the `probes` dictionary, running the case, and postprocessing. Both post-/preprocessing files are written in the Python language, and the resulting plots will be saved in the `figures` directory. The evolution of overall variables is depicted in Fig. 4.2. Fig. 4.3 portrays the particle size distribution based on the number of agglomerates, N_{agg}, and the number of primary particles, N_{pri}.

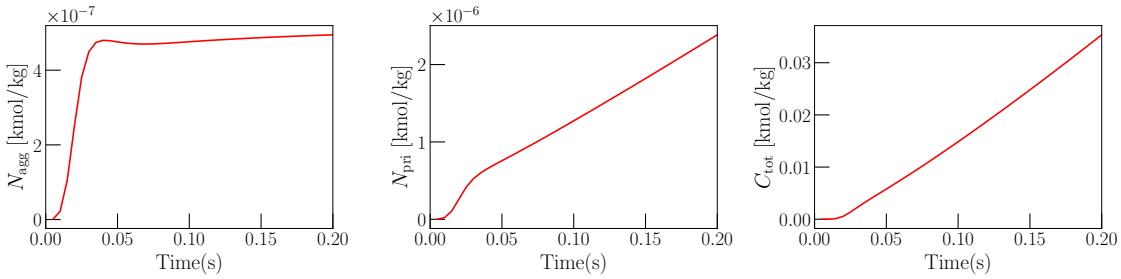

Figure 4.2: Evolution of various soot variables over time.

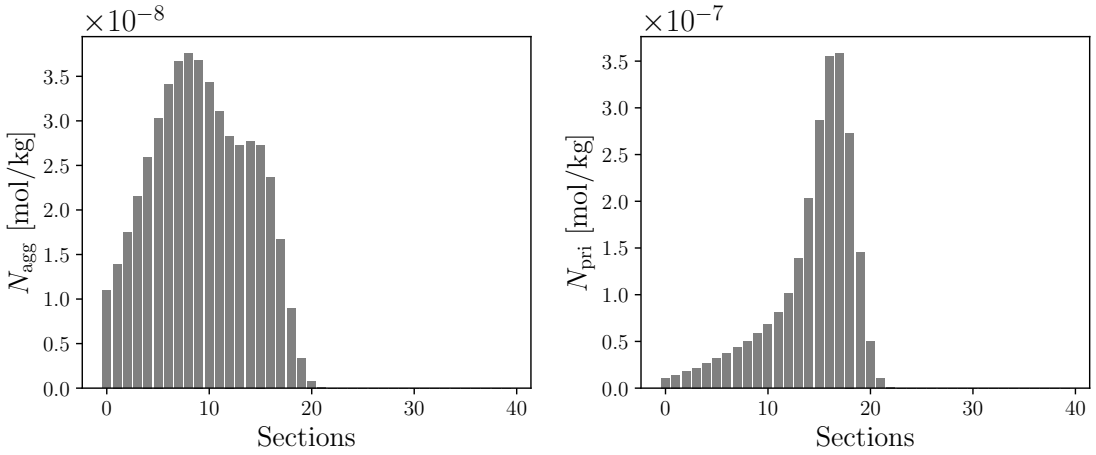

Figure 4.3: Soot particle size distribution at t=0.2 (s)

.

Bibliography

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

[2] D. Parkinson, "The reinforcement of rubber by carbon black," *British Journal of Applied Physics*, vol. 2, no. 10, p. 273, 1951.

[3] A. Y. Watson and P. A. Valberg, "Carbon black and soot: two different substances," *AIHAJ-American Industrial Hygiene Association*, vol. 62, no. 2, pp. 218–228, 2001.

[4] L. Pfaffmann, M. Müller, W. Bauer, F. Scheiba, S. Jaiser, M. Baunach, P. Scharfer, and H. Ehrenberg, "Analysis of the binder and carbon black distribution in graphite electrodes for lithium ion batteries using electron dispersive x-r ay spectroscopy and energy selective backscatter electrons," in *European Microscopy Congress 2016: Proceedings*, pp. 844–845, Wiley Online Library, 2016.

[5] A. D'Anna, "Combustion-formed nanoparticles," *Proceedings of the Combustion Institute*, vol. 32, no. 1, pp. 593–613, 2009.

[6] H. Wang, "Formation of nascent soot and other condensed-phase materials in flames," *Proceedings of the Combustion institute*, vol. 33, no. 1, pp. 41–67, 2011.

[7] S. H. Chung and A. Violi, "Insights on the nanoparticle formation process in counterflow diffusion flames," *Carbon*, vol. 45, no. 12, pp. 2400–2410, 2007.

[8] M. Frenklach, "Reaction mechanism of soot formation in flames," *Physical chemistry chemical Physics*, vol. 4, no. 11, pp. 2028–2037, 2002.

[9] 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.

[10] R. A. Dobbins, "Hydrocarbon nanoparticles formed in flames and diesel engines," *Aerosol Science and Technology*, vol. 41, no. 5, pp. 485–496, 2007.

[11] Q. Zhang, M. J. Thomson, H. Guo, F. Liu, and G. J. Smallwood, "Modeling of oxidation-driven soot aggregate fragmentation in a laminar coflow diffusion flame," *Combustion science and technology*, vol. 182, no. 4-6, pp. 491–504, 2010.

[12] G. A. Kelesidis, *Morphology and optical properties of flame-made nanoparticles.* PhD thesis, ETH Zurich, 2019.

[13] G. A. Kelesidis, E. Goudeli, and S. E. Pratsinis, "Flame synthesis of functional nanostructured materials and devices: Surface growth and aggregation," *Proceedings of the Combustion Institute*, vol. 36, no. 1, pp. 29–50, 2017.

[14] M. R. Kholghy and G. A. Kelesidis, "Surface growth, coagulation and oxidation of soot by a monodisperse population balance model," *Combustion and Flame*, vol. 227, pp. 456–463, 2021.

[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] S. Tsantilis and S. E. Pratsinis, "Evolution of primary and aggregate particle-size distributions by coagulation and sintering," *AIChE Journal*, vol. 46, no. 2, pp. 407–415, 2000.

[17] D. R. Handwerk, P. D. Shipman, C. B. Whitehead, S. Ozkar, and R. G. Finke, "Mechanism-enabled population balance modeling of particle formation en route to particle average size and size distribution understanding and control," *Journal of the American Chemical Society*, vol. 141, no. 40, pp. 15827–15839, 2019.

[18] M. J. Thomson, "Modeling soot formation in flames and reactors: Recent progress and current challenges," *Proceedings of the Combustion Institute*, 2022.

[19] Q. Zhang, *Detailed modeling of soot formation/oxidation in laminar coflow diffusion flames*. University of Toronto Toronto, Canada, 2009.

[20] M. Adib, "Implementation of a monodisperse population balance model in laminar combustion model," in *Proceedings of CFD with OpenSource Software* (H. Nilsson, ed.), 2022. http://dx.doi.org/10.17196/OSCFD#YEAR2022.

[21] A. Veshkini, *Understanding Soot Particle Growth Chemistry and Particle Sizing Using a Novel Soot Growth and Formation Model*. University of Toronto (Canada), 2015.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

Study Questions

1. Why is there no need to specify a boundary condition for C_{tot} in `laminarSPBM`, while it is necessary for `laminarSoot`?
2. What is the impact of each mechanism (inception, HACA, and PAH surface growth, oxidation, and coagulation) on the soot particle size distribution?
3. Why is the data type used for storing inception data different from that of other mechanisms?
4. What information is received from the OpenFOAM chemistry solver?
5. Which function takes care of transferring particles between sections?

Appendix A

Chemical Source Terms

This appendix is a part of `laminarSoot` documentation [20].

A.1 Inception

The inception is described using reactive dimerization of polycylic aromatic hydrocarbons (PAHs) [22] 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

$$\mathrm{PAH}_i + \mathrm{PAH}_j \underset{\mathrm{k_{REV}}}{\overset{\mathrm{k_{FWD}}}{\rightleftarrows}} \mathrm{Dimer}^*_{\mathrm{ij}}, \tag{A.1}$$

$$\mathrm{Dimer}^*_{\mathrm{ij}} \xrightarrow{\mathrm{k_{REAC}}} \mathrm{Dimer}_{\mathrm{ij}}. \tag{A.2}$$

In Equation (A.1), k_{FWD} is the forward rate of physical dimerization and computed as

$$k_{FWD} = 2.2 \cdot 0.1 \cdot Av \cdot d_{ij}^2 \sqrt{\frac{8\pi k_B T}{m_{ij}}}, \tag{A.3}$$

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 [23] using

$$m_{PAH} = \frac{W_{PAH}}{Av}, \tag{A.4}$$

$$\rho_{PAH} = 171943.5197 \frac{n_{C,PAH} W_{carbon} + n_{H,PAH} W_{hydrogen}}{n_{C,PAH} + n_{H,PAH}}, \tag{A.5}$$

$$V_{PAH} = \frac{m_{PAH}}{\rho_{PAH}}, \tag{A.6}$$

$$d_{PAH} = \left(\frac{6V_{PAH}}{\pi}\right)^{1/3}. \tag{A.7}$$

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

$$k_{REV} = k_{FWD} 10^{-b} e^{-a\epsilon ln(10)/(RT)}, \tag{A.8}$$

$$\epsilon = cW_{ij} - d, \tag{A.9}$$

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 [24]) and b=1.8, c=933420 j/kg, and d=34053 j/mol [22].

The rate of chemical bond formation, k_{REAC} is defined in the Arrhenius form [25] as

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

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

$$\omega_{dimer_{ij}} = k_{REAC} \frac{k_{FWD}[PAH_i][PAH_j]}{k_{REV} + k_{REAC}}. \tag{A.11}$$

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

$$\omega_{PAH_i} = \omega_{PAH_j} = -\omega_{dimer_{ij}}. \tag{A.12}$$

Therefore, the carbon content that transforms from the gas to the solid phase through the inception process can be computed by

$$S_{C_{\mathrm{tot,inception}}} = \sum_{i=1}^{n} \sum_{j=i}^{n} C_{ij} \omega_{dimer_{ij}}, \tag{A.13}$$

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.

A.2 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 [9] is described by a set of elementary with given rates that are listed in Table A.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

$$\omega^i_{c_2h_2} = \alpha k_{f4} [\mathrm{C_2H_2}][\mathrm{C_{soot}{}^\circ}]^i, \tag{A.14}$$

$$\frac{d}{dt}[\mathrm{C_2H_2}] = -\omega^i_{c_2h_2}. \tag{A.15}$$

In Equation (A.14), k_{f4} refers to forawrd reaction rate constant of 4th reaction in reaction 4 in Table A.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 the number of arm-chair and zig-zag hydrogenated sites on soot particle [26] using

$$S^{C,i}_{grow|HACA} = 2\omega^i_{c_2h_2}/\rho, \tag{A.16}$$

In Equation (A.14), α is the surface reactivity of soot defined by an empirical relation [9] as

$$\alpha = \tanh\left(\frac{12.56 - 0.00563 \cdot T}{\log_{10}\left(\frac{\rho_{soot}\frac{\pi}{6}(d^i_p)^3 \cdot Av}{W_{carbon}}\right)} - 1.38 + 0.00068 \cdot T\right). \tag{A.17}$$

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

$$[\mathrm{C_{soot}{}^\circ}]^i = A^i_{tot} \frac{\rho}{Av} \chi_{soot^\circ}. \tag{A.18}$$

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

$$A^i_{tot} = N^i_{pri} Av \cdot \pi (d^i_p)^2, \tag{A.19}$$

Table A.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

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

$$\chi_{soot^\circ} = \frac{k_{f1}[H] + k_{f2}[OH]}{k_{r1}[H_2] + k_{r2}[H_2O] + k_{f3}[H] + k_{f4}[C_2H_2] + k_{f5}[O_2] + k_{f1}[H] + k_{f2}[OH]} \chi_{soot_{CH}}, \quad \text{(A.20)}$$

where $\chi_{soot_{CH}} = 2.3 \times 10^{19} m^{-2}$. In Equation (A.20), k_{r1} denotes the reverse rate of the first reaction in Table A.1, and the rest of the reaction rates follow the same naming convention.

A.3 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

$$PAH + Soot \underset{k_{rv,ad}}{\overset{k_{fw,ad}}{\rightleftarrows}} Soot{-}PAH^*, \quad \text{(A.21)}$$

$$Soot{-}PAH^* \xrightarrow{k_{rc,ad}} Soot{-}PAH. \quad \text{(A.22)}$$

The forward rate of physical adsorption, $k_{fw,ad}$, in Equation (A.21) is computed by harmonic mean of collision frequency of soot particles and PAH molecules in free molecular and continuum regime as

$$k_{fw,ad} = \frac{\beta_{fm,ad} \cdot \beta_{cont,ad}}{\beta_{fm,ad} + \beta_{cont,ad}} Av, \quad \text{(A.23)}$$

where $\beta_{fm,ad}$ is obtained [25] as

$$\beta_{fm,ad} = 2.2 \sqrt{\frac{\pi k_B T}{2} \left(\frac{1}{m^i_{agg}} + \frac{1}{m_{PAH}} \right)} \left(d^i_g + d_{PAH} \right)^2, \quad \text{(A.24)}$$

where $m^i_{agg} = C^i_{tot} \cdot W_{carbon} / (N^i_{agg} \cdot Av)$ is the mass of soot agglomerate. $\beta_{cont,ad}$ is computed by

$$\beta_{cont,ad} = \frac{2 k_B T}{3\mu} \left[\frac{C_s(d^i_m)}{d^i_g} + \frac{C_s(d_{PAH})}{d_{PAH}} \right] \left(d^i_g + d_{PAH} \right), \quad \text{(A.25)}$$

$$C_s(d) = 1 + \frac{2\lambda}{d} \left[1.21 + 0.4 e^{(-0.78 d/\lambda)} \right] \left(d^i_g + d_{PAH} \right). \quad \text{(A.26)}$$

The reverse rate of physical adsorption, $k_{rv,ad}$, is computed similar to reverse physical inception rate (Equation (A.8)) as

$$k_{rv,ad} = k_{fw,ad} \times 10^{-b} e^{-a\epsilon ln(10)/(RT)}, \quad \text{(A.27)}$$

$$\epsilon = c\frac{MW_{soot} \cdot MW_{PAH}}{MW_{soot} + MW_{PAH}} - d, \tag{A.28}$$

where $\mathrm{MW_{soot}} = \mathrm{C_{tot}} \cdot \mathrm{W_{carbon}}/\mathrm{N_{agg}}$ is the equivalent molecular weight of soot, and a, b, c, and d have the same values as Equation (A.8).

The rate of chemical adsorption, $\mathrm{k_{rc,ad}}$ is defined in the Arrhenius form [25] as

$$k_{rc,ad} = 2 \times 10^{10} \cdot e^{(-96232/RT)}. \tag{A.29}$$

The total adsorption rate can be calculated assuming a steady-state concentration for physically adsorbed PAH on soot, $\partial[\mathrm{Soot} - \mathrm{PAH^*}]/\partial \mathrm{t} = 0$, similar to inception rate (Equation (A.11)) as

$$\omega^i_{pah,ad} = k_{rc,ad}\frac{k_{fw,ad}[\mathrm{Soot}]^i[\mathrm{PAH}]}{k_{rv,ad} + k_{rc,ad}}, \tag{A.30}$$

$$[\mathrm{Soot}]^i = \rho N^i_{agg}. \tag{A.31}$$

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

$$S^{C,i}_{grow|ad} = \sum_{k=1}^{n} C_{PAH,k} \cdot \omega^i_{pah,ad,k}, \tag{A.32}$$

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

$$\omega^i_{PAH,k} = -\omega^i_{pah,ad,k}. \tag{A.33}$$

A.4 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

$$\omega^i_{o2} = \alpha k_{f5}[\mathrm{O_2}][\mathrm{C_{soot^\circ}}]^i, \tag{A.34}$$

$$\omega^i_{oh} = \alpha k_{f6}[\mathrm{OH}]N^i_{agg}\rho. \tag{A.35}$$

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

$$S^{C,i}_{ox} = -(2\omega^i_{o2} + \omega^i_{oh})/\rho, \tag{A.36}$$