Cantera-soot handbook

sectional soot computations within Cantera

Authors: **E. Lameloise**

Contents

1 Introduction

Cantera-soot is a modified version of Cantera/2.3-avbp that enables the computation of soot for axisymmetric stagnation flows (impinging jets, burner and counterflow 1D flames).

1.1 Presentation

Cantera-soot uses a Discrete Sectional Method (DSM) to solve the soot particles distribution alongside the flame gaseous chemistry. In comparison with its predecessor (CAN2SOOT), Cantera-soot is able to take into account the consumption of gaseous species by the soot particles. Moreover, Cantera-soot is also able to solve soot radiative heat transfer using an optically-thin limit approximation.

Most of the equations that are implemented and introduced in the following were taken from the PhD. thesis of Pedro Rodrigues [11]

The basic usage of Cantera is not affected by the implementation of the soot module. Additionnal capabilities of the code (soot calculation) are activated through the keyword *sections* when building the *Flame* Cantera object.

In the following, the implemented equations are introduced. Built-in API methods are then listed. Eventually, two examples are treated : a burner and an impinging-jet flame that were taken from the ISF 4 workshop [10].

1.2 Installation

Cantera-soot is currently available on CERFACS private Gitlab server (Nitrox) as the WIP/SOOT branch of the Cantera-avbp project. It can be cloned through :

```
git clone -b WIP/SOOT git@gitlab.com:cerfacs/chemistry/cantera-avbp.git
It will hopefully soon be available on CERFACS public gitlab server \cite{CERFACSgitl
```

It can be installed on both MacOS and Linux platforms using the tutorial available on CERFACS chemistry webpage [3].

2 Sectional method

2.1 Population Balance Equation

It is assumed that the volume distribution of the soot population's particle density n $[cm^{-6}]$ follows the Population Balance Equation (PBE) (1) :

$$\frac{\partial n}{\partial t} + \underbrace{\nabla \cdot (\mathbf{u}n)}_{\text{Convection}} - \underbrace{\nabla \cdot \left(C_{th} \nu \frac{\nabla T}{T} n \right)}_{\text{Thermophoresis}} = \underbrace{\nabla \cdot (D_s \nabla n)}_{\text{Diffusion}} + \underbrace{\dot{n}_s}_{\text{Source}} \tag{1}$$

Because of the complex chemistry that governs soot formation, soot particles exist in a wide range of size covering approx. 1 nm to 100 nm. Accurately describing the distribution of particles in terms of size at each grid point thus proves complex and computationally expensive (usually involving stochastic methods such as Monte-Carlo appoaches). In order to reduce the computational costs while keeping an accurate enough description of the soot particles distribution, the soot distribution function is discretized in different groups with the assumption of constant volume fraction density ($q(v) = vn(v) = q_k\ [cm^{-3}],\ v \in [V_k^{min}, V_k^{max}]$) within each group. Each group might be interpreted as the lump of any soot particle in its volume range and is called a section.

2.2 Volumes creation

Volumes are created to follow a geometrical progression :

$$\begin{aligned} V_k^{max} &= V^{MIN} \left(\frac{V^{MAX}}{V^{MIN}} \right)^{\frac{k}{N_s}} \\ V_k^{min} &= V_{k-1}^{max} \end{aligned} \tag{2}$$

V^{MIN} is computed as the volume of the smallest possible soot particle. As soot stems from the collision of two so-called dimers - each dimer resulting of the collision of two precursors (Policylic Aromatic Hydrocarbon (PAH)) -, the minimal volume equals four times the volume of the smallest precursor. The maximal volume is the volume of the biggest expected soot particle, taken from experimental data.

$$\begin{aligned} V^{MIN} &= 2 \min(n_k^C) V_{C_2} \\ V^{MAX} &= 1.10^{-9}\ cm^3 \end{aligned} \tag{3}$$

With n_k^C the number of carbon atoms in the considered precursors. The volume of a C-C binding in solid phase is computed as :

$$V_{C_2} = 2 \frac{W_C}{\rho_s N_A} = 2.14\ 10^{-23}\ cm^3 \tag{4}$$

Where $\rho_s = 1.86\ 10^3 g.cm^{-3}$ is the density of soot, considered constant whatever the size of the particle.

Two conditions must be accounted for to compute the maximal volume of the first section :

- The first section should host all the nascent soot particles :

$$V_1^{max} \geq 2\max(n_k^C)V_{C_2}$$

- The computation of condensation (see Sect.5.3)involves an integration over the interval $[V_k^{max} - V_d, V_k^{min}]$. For the interval to remain positive :

$$V_1^{max} \geq V_1^{max} + \underbrace{V_d}_{\leq \max(n_k^C)V_{C_2}}$$

Which gives the overall condition :

$$V_1^{max} = V_{C_2} \max\left(2\min(n_k^C) + \max(n_k^C), 2\max(n_k^C)\right) \tag{5}$$

Sections mean volume can then be computed as the logarithmic average of the maximal and mean volumes. Mean diameters computation is then straighforward :

$$\begin{cases} V_k^{mean} & = \frac{V_k^{max} - V_k^{min}}{\log\left(V_k^{max}/V_k^{min}\right)} \\ d_k^{mean} & = \left(\frac{6V_k^{mean}}{\pi}\right)^{1/3} \end{cases} \tag{6}$$

In order to compute mean cross-section, soot aggregates fractality must be taken into account. Soot aggregates fractal dimension was found to be approximately equal to $D_f = 1.8$ for particles volumes bigger than $V_{morpho} = 3.98\ 10^{-19}\ cm^3$.

Mean cross-section and collisional diameter are then computed as :

$$\begin{cases} d_k^{coll} & = \frac{6}{(36\pi)^{1/D_f}} V_k^{mean(D_f-2)/D_f} \left(S_{C_2}(V_k^{mean}/V_{C_2})^{\theta_k/3}\right)^{(3-D_f)/D_f} \\ S_k^{mean} & = S_{C_2}\left(\frac{V_k^{mean}}{V_{C_2}}\right)^{\theta_k/3} \end{cases} \tag{7}$$

Where

$$\theta_k = \min\left(2.0, \frac{3\log(V_k^{mean}/V_{morphology}) + 2\log(V_{morphology}/V_{C_2})}{\log(V_k^{mean}/V_{C_2}}\right)\right) \tag{8}$$

3 Transport equation

As the distribution was dicretized in N_s sections, N_s different variables must be transported. It was chosen to solve this system alongside the usual system of equations solved by CANTERA, meaning that CANTERA-soot solves a system of $4 + n_{\text{species}} + N_s$ equations.

The PBE (eq. 1) is discretized with the assumption of constant volume fraction density q_k within each section k. Moreover, as gaseous species are solved in terms of mass fraction by CANTERA, it is chosen to solve the soot mass fraction Y_s instead of the soot particles density volume distribution n. The 1D discretized PBE implemented in CANTERA-soot is thus :

$$\underbrace{\rho v \frac{\partial Y_{s,k}}{\partial z}}_{\text{Convection}} - \underbrace{\frac{\partial}{\partial z}\left(C_{th}\mu\frac{1}{T}\frac{\partial T}{\partial z}Y_{s,k}\right)}_{\text{Thermophoresis}} = \underbrace{\frac{\partial}{\partial z}\left(\rho D_{s,k}\frac{\partial Y_{s,k}}{\partial z}\right)}_{\text{Diffusion}} + \underbrace{\rho_s \rho \dot{q}_{s,k}}_{\text{Source}} \tag{9}$$

In CANTERA, the Jacobian matrix is computed out of the matrix of residuals. The calculation of each residual for soot at point j and for section k is :

$$\begin{aligned}\text{rsd}_{k,j} =& \frac{\text{Source} - \text{Convection} + \text{Diffusion} + \text{Thermophoresis}}{\rho} \\ & - \frac{Y_{s,k,j}(t) - Y_{s,k,j}(t-1)}{\Delta t}\end{aligned} \tag{10}$$

3.1 Convective fluxes

Convective fluxes are computed through an upwind scheme on the derivative of the soot mass fraction. It was mainly chosen because it is already implemented to compute the convective fluxes of the gaseous species.

$$\text{convection}_{s,k,} = \begin{cases} \rho_j u_j \frac{Y_{s,k,j} - Y_{s,k,j-1}}{z_j - z_{j-1}} & u \geq 0 \\ \rho_j u_j \frac{Y_{s,k,j} - Y_{s,k,j+1}}{z_j - z_{j+1}} & u < 0 \end{cases} \tag{11}$$

3.2 Diffusive fluxes

Soot diffusive mass fluxes are defined according to eq. 9 :

$$j_{s,k} = \rho D_{s,k} \frac{\partial Y_{s,k}}{\partial z} \tag{12}$$

Soot particles diffusion was studied by Epstein [5] and the soot particles diffusion coeffient can be expressed as :

$$D_{s,k} = \frac{3}{2\rho} \left(1 + \frac{\alpha_T \pi}{8}\right)^{-1} \frac{1}{d_{mean,k}^2} \sqrt{\frac{W_{\text{gas}} k_b T}{2\pi N_A}} \tag{13}$$

Where α_T is a thermal accodomodation factor usually taken as 0.9

As diffusion and diffusive mass fluxes are diffusive terms, they are computed through centered differences. Diffusion of soot particles from section k at point j can thus be written :

$$\text{Diffusion}_{k,j} = \frac{2}{z_{j+1} - z_{j-1}} \left(j_{s,k,j+\frac{1}{2}} - j_{s,k,j-\frac{1}{2}}\right) \tag{14}$$

Where diffusive mass flux are evaluated at midpoints $j \pm \frac{1}{2}$:

$$j_{s,k,j\pm\frac{1}{2}} = \rho_{j\pm\frac{1}{2}} D_{s,k,j\pm\frac{1}{2}} \frac{Y_{s,k,j\pm\frac{1}{2}+\frac{1}{2}} - Y_{s,k,j\pm\frac{1}{2}-\frac{1}{2}}}{z_{j\pm\frac{1}{2}+\frac{1}{2}} - z_{j\pm\frac{1}{2}-\frac{1}{2}}} \tag{15}$$

Density and mean molecular weight are evaluated at midpoint by averaging the two adjacent points. CANTERA provides built-in *"MeanMolecularWeight"* and *"Density"* methods in the *"Thermo"* class but it was remarked that setting the gas state to each midpoint and pointing towards the values was too time-expensive.

3.3 Thermophoresis

As well as diffusion, thermophoresis is a diffusive term that involves second order derivatives. Each derivative is computed through centered differences. The first one is computed at midpoints $j \pm \frac{1}{2}$ while the second one is computed at point j. Thermophoretic diffusion thus writes :

$$\text{Thermophoresis}_{k,j} = \frac{2}{z_{j+1} - z_{j-1}} \left(j^{\text{thermoph}}_{s,k,j+\frac{1}{2}} - j^{\text{thermoph}}_{s,k,j-\frac{1}{2}} \right) \tag{16}$$

With the thermophoretic mass flux computed in the free molecular regime [4] :

$$j^{\text{thermoph}}_{s,k,j\pm\frac{1}{2}} = \underbrace{C_{th}}_{0.554} \mu \frac{1}{T} \left. \frac{\partial T}{\partial z} \right|_{j\pm\frac{1}{2}} \tag{17}$$

Where the temperature gradient is once again computed through centered differences :

$$\left. \frac{\partial T}{\partial z} \right|_{j+\frac{1}{2}} = \frac{T_{j+1} - T_j}{z_{j+1} - z_j} \tag{18}$$

Viscosity is evaluated at midpoint by a Sutherland law. Indeed, CANTERA provides a built-in *"viscosity"* method in the *"Transport"* class but it was remarked that setting the gas state to each midpoint and pointing towards the value of the viscosity was too time-expensive.

4 Radiative heat transfers

Radiative heat transfers are taken into account through RDG theory. The radiating power P_s^R of soot particles is computed as :

$$P_s^R = -4\sigma\kappa_{Pl,s}\tilde{T}^4 \tag{19}$$

Where $\kappa_{Pl,s}$ is the Planck mean absorption coefficient (Modest [8]), calculated as :

$$\kappa_{Pl,s}(T) = 3.83 \frac{C_0 f_V T}{C_2} \tag{20}$$

With f_V the soot volume fraction defined as

$$f_V = \rho/\rho_s \sum_{k=1}^{N_s} Y_{s,k} \tag{21}$$

C_0 and C_2 the Planck constants :

$$\begin{cases} C_0 &= \frac{36\pi ab}{(a^2-b^2+2)+4a^2b^2} \\ C_2 &= \frac{hc}{k_b} \end{cases} \tag{22}$$

Where a and b are respectively the real and immaginary part of the refractive index of soot (Smyth and Shaddix [12]) : $m = a - ib = 1.57 - 0.56i$, $h \simeq 6.626\ 10^{-34}\ J.s^{-1}$ and $c \simeq 3.0\ 10^8\ m.s^{-1}$ is the speed of light in void.

The radiative power of soot particles is then added to the radiative power of the gaseous species considered by CANTERA (H_2O and CO_2).

Note that soot related radiative heat losses are only computed if energy equation and gaseous radiative heat losses are enabled (*energy_enabled=True* and *radiation_enabled=True*).

5 Source terms

The theory used to compute source terms is taken from Blanquart & Pitsch [1] (in Bockhorn [2]) and Mueller [9].

5.1 Dimerization

The dimerization corresponds to the coalescence of two PAH particles, resulting in a dimer. The overall dimerization rate r_d $[g.s^{-1}]$ is computed as the sum of each PAH dimerization rate $K_{00,k}$ $[mol.g.s^{-1}]$:

$$\begin{aligned} r_d &= \sum_{k=1}^{N_{PAH}} K_k^{00} N_A \\ K_k^{00} &= \frac{1}{\rho} C_n W_k^4 \sqrt{\frac{4\pi k_b T}{\frac{W_C}{N_A} n_k^C}} [PAH_k]^2 d_k^2 N_A \end{aligned} \tag{23}$$

Where $C_n = 7.5\ 10^{-12}\ mol^4.g^{-4}$ and d_k is the diameter of the considered PAH.

The volume of the resulting dimer V_d $[cm^3]$ is computed by averaging each production rate K_k^{00} weighed by the number of carbon atoms in the corresponding PAH n_k^C and multiplying it by the volume of a C-C binding V_{C_2} :

$$V_d = \frac{\sum_{k=1}^{N_{PAH}} K_k^{00} n_k^C}{\sum_{k=1}^{N_{PAH}} K_k^{00}} V_{C_2} \tag{24}$$

The consumption of each PAH by the dimerization process is taken into account in sootConsumption ($[mol.cm^{-3}.s^{-1}]$) :

$$\text{sootConsumption}_k = 2K_k^{00}\rho \tag{25}$$

The factor 2 is due to the fact that a single mole of dimer is created by the collision of two moles of PAHs. This array gathers the consumption of any species by soot particles. That's why it will be further modified when computing surface chemistry.

5.2 Nucleation

Nucleation is the inception of a primary soot particle by the coalescence of two dimers. As proposed by Blanquart and Pitsch [1], dimers are lumped into a single equivalent dimer

during the dimerization process, thus a single collision process between two equivalent dimers occurs during nucleation. The dimers collision frequency β_d $[cm^6.s^{-1}.g^{-1}]$ is given in by :

$$\beta_d = \varepsilon_{nucl}\beta_{fm}\sqrt{\frac{2}{V_d}}\left(2\left(\frac{6V_d}{\pi}\right)^{1/3}\right)^2 \tag{26}$$

Whera $\varepsilon_{nucl} = 2.5$ is an efficiency coefficient to take Van der Walls interactions into account and $\beta_{fm} = \sqrt{\frac{\pi k_b T}{2\rho_s}}\frac{1}{\rho}$ $[cm^{\frac{11}{2}}.s^{-1}.g^{-1}]$ is a term that appears each time collision occurs in free molecular regime.

The dimer particles density N_d $[cm^{-3}]$ is computed through :

$$N_d = \sqrt{\frac{r_d}{\beta_d}} \tag{27}$$

Eventually, the nucleation source term q_1^{nucl} $[cm^3.g^{-1}.s^{-1}]$ is computed for the first section as follows:

$$\dot{q}_1^{nucl} = V_d\beta_d N_d^2 \tag{28}$$

This therm will be further corrected. Indeed, dimers can either create a new soot nuclei or condensate onto an already existing soot particle. Thus, each dimer particle cannot account for nucleation.

5.3 Condensation

Condensation correponds to the collision between a dimer and a soot particle. The dimer condensates on the soot particle, thus increasing its mass and volume. The overall condensation source term k_{cond} $[m^3.s^{-1}.kg^{-1}]$ is computed as the sum of each section's source term k_k^{cond} :

$$\begin{aligned}
k_{cond} &= \sum_{k=1}^{N_s} k_k^{cond} \\
k_k^{cond} &= \beta_k^{cond} q_k \frac{V_k^{max} - V_k^{min}}{V_k^{mean}} \\
\beta_k^{cond} &= \varepsilon_{cond}\beta_{fm}\sqrt{\frac{1}{V_k^{mean}} + \frac{1}{V_d}} + \left(d_k^{coll} + \left(\frac{6V_d}{\pi}\right)^{1/3}\right)^2
\end{aligned} \tag{29}$$

β_k^{cond} is frequency of the collisions between a dimer and a particle of section k. The efficiency coefficient is taken as $\varepsilon_{cond} = 1.3$.

The dimer particle density available for nucleation can now be corrected :

$$\begin{aligned}
N_d &= -\frac{k_{cond}}{2\beta_d} + \sqrt{\frac{r_d}{\beta_d} + \left(\frac{k_{cond}}{2\beta_d}\right)^2} \\
\dot{q}_1^{nucl} &= V_d\beta_d N_d^2
\end{aligned} \tag{30}$$

For each section, condensation source term is then divided into three different terms depending whether :

- the mass comes from a dimer that leaves the gas phase towards section k (term_k^1),

$$\begin{cases} \text{term}_k^1 & = N_d \beta_k^{cond} q_k \frac{V_d \Delta V_k}{V_k^{moy}} \\ V_k^{moy} & = \frac{(V_k^{max} - V_d) + V_k^{min}}{2} \\ \Delta V_k & = (V_k^{max} - V_d) - V_k^{min} \end{cases} \tag{31}$$

- the mass comes from a dimer that condensates onto a particle of section $k-1$ to enter section k (term_k^2),

$$\begin{cases} \text{term}_k^2 & = N_d \beta_{k-1}^{cond} q_{k-1} \frac{V_d \Delta V_{k-1}}{V_{k-1}^{moy}} \\ V_k^{moy} & = \frac{V_k^{max} + (V_k^{max} - V_d)}{2} \\ \Delta V_k & = V_k^{max} - (V_k^{max} - V_d) = V_d \end{cases} \tag{32}$$

- the mass comes from a particle that leaves section $k-1$ towards section k through condensation(term_k^3).

$$\begin{cases} \text{term}_k^3 & = N_d \beta_{k-1}^{cond} q_{k-1} \frac{V_{k-1}^{moy} \Delta V_{k-1}}{V_{k-1}^{moy}} \\ V_k^{moy} & = \frac{V_k^{max} + (V_k^{max} - V_d)}{2} \\ \Delta V_k & = V_k^{max} - (V_k^{max} - V_d) = V_d \end{cases} \tag{33}$$

Where the terms V_{moy} and ΔV_k are due to an integration of the PSDF within each section (details are given by P. Rodrigues [11]).

Source terms are then computed for each section as :

$$\dot{q}_{cond}(k) = \underbrace{\text{term}_k^1}_{\substack{\text{gas} \to k \\ \text{dimer}}} + \underbrace{\text{term}_k^2}_{\substack{k-1 \to k \\ \text{dimer}}} + \underbrace{\text{term}_k^3}_{\substack{k-1 \to k \\ \text{soot}}} - \underbrace{\text{term}_{k+1}^3}_{\substack{k \to k+1 \\ \text{soot}}} \tag{34}$$

5.4 Coagulation

The coagulation occurs when two soot aggregates merge to form a bigger aggregate.

The overall coagulation rate of two particles from sections (l,k) - written $\beta_{l,k}^{coag}$ - is the mix of two collision regimes : the free molecular ($\beta_{l,k}^{fm}$) and the continous ($\beta_{l,k}^{cont}$) ones :

$$\begin{cases} \beta_{l,k}^{fm} & = \varepsilon_{coag} \beta_{fm} \sqrt{\frac{1}{V_k^{mean}} + \frac{1}{V_l^{mean}}} \left(d_k^{coll} + d_l^{coll} \right)^2 \\ \beta_{l,k}^{cont} & = \frac{2 k_b T}{3 \mu} \left(d_k^{coll} + d_l^{coll} \right) \left(\frac{Cu_k}{d_k^{coll}} + \frac{Cu_l}{d_l^{coll}} \right) \end{cases} \tag{35}$$

Where Cu_k & Kn_k are respectively the Cunningham and Knudsen numbers :

$$\begin{cases} Cu_k & = 1 + 1.257 Kn_k \\ Kn_k & = \frac{2 \lambda_{gas}}{d_k^{coll}} \end{cases} \tag{36}$$

The mean free path (λ_{gas}) is defined by the kinetic theory :

$$\lambda_{gas} = \frac{k_b T}{\sqrt{2} \pi d_{gaz}^2 P} \ , \ d_{gaz} = 2.0 \ 10^{-8} \ cm \tag{37}$$

The overall coagulation rate is then computed as :

$$\beta_{l,k}^{coag} = \frac{\beta_{l,k}^{fm}\beta_{l,k}^{cont}}{\beta_{l,k}^{fm} + \beta_{l,k}^{cont}} \tag{38}$$

It can be noted that $\beta_{l,k}^{coag} \simeq \min\left(\beta_{l,k}^{fm}, \beta_{l,k}^{cont}\right)$. However, this approximation is not used in CANTERA-soot.

Once the overall coagulation rate of particles from (l,k) is computed, the source terms relative to each section can be computed. These terms correspond to the mass that leaves both section through coagulation.

$$\begin{cases} \text{term}_k &= \beta_{l,k}^{coag} q_l q_k \frac{\Delta V_l \Delta V_k}{V_k^{mean}} \\ \text{term}_l &= \beta_{l,k}^{coag} q_l q_k \frac{\Delta V_l \Delta V_k}{V_l^{mean}} \end{cases} \tag{39}$$

Where $\Delta V_k = V_k^{max} - V_k^{min}$ is the size of the section.

The minimal size of the newly formed particle is equal to $V_{min,l} + V_{min,k}$, which is said to be in section m. Section m is thus chosen so that $V_l^{min} + V_k^{min} \in [V_m^{min}, V_m^{max}]$.

Particles created by the coagulation of particles from (l,k) will thus at least enter section m :

$$\text{if } V_k^{max} + V_l^{max} \leq V_m^{max} \ , \ \begin{cases} term_m &= term_l + term_k \\ term_{m+1} &= 0 \end{cases} \tag{40}$$

Where $term_m$ & $term_{m+1}$ are respectively the source terms for section m and $m+1$ resulting from the coagulation of (l,k).

If the newly formed particle can be bigger than V_m^{max}, some mass will enter section $m+1$ (note that it is not possible to reach section $m+2$ because of the way the volumes of the sections were chosen). In this case :

$$\text{if } V_k^{max} + V_l^{max} > V_m^{max} \ , \ \begin{cases} term_m &= (term_l + term_k) \frac{V_m^{max} - V_k^{max} - V_l^{max}}{\Delta V_l + \Delta V_k} \\ term_{m+1} &= (term_l + term_k) \frac{V_k^{max} + V_l^{max} - V_m^{max}}{\Delta V_l + \Delta V_k} \end{cases} \tag{41}$$

The overall source terms are then evaluated as a sum of the contribution of each sections :

$$\begin{cases} \dot{q}_k^{coag} &= -\sum_{k=1}^{Ns} \text{term}_k \\ \dot{q}_l^{coag} &= -\sum_{l=1}^{Ns} \text{term}_l \\ \dot{q}_m^{coag} &= \sum_{m=1}^{Ns} \text{term}_m \\ \dot{q}_{m+1}^{coag} &= \sum_{m=1}^{Ns-1} \text{term}_{m+1} \end{cases} \tag{42}$$

5.5 Surface chemistry

Surface chemistry corresponds to the reactions between the soot aggregates and the gaseous media. In opposition to the condensation, the considered gaseous species are not the precursors. In our case, considered species are : H_2, OH, H_2O, C_2H_2, O_2 & H.

The chosen reactions are those from the HACA-RC mechanism. They are summed up in Tab. 1. The reactions follow an Arrhenius law :

$$k_i = A_i T^{\beta} \exp\left(\frac{-Ea}{RT}\right), \text{ with } \begin{cases} k_i & : \text{ Rate constant } [s^{-1} \text{ or } m^3.mol^{-1}] \\ A & : \text{ Pre-exponential factor } [s^{-1} \text{ or } m^3.mol^{-1}] \\ \beta & : \text{ Fudge factor } [-] \\ Ea & : \text{ Activation energy } [kJ.mol^{-1}] \end{cases} \tag{43}$$

This set of reactions was proposed by Mauss [7] as well as the Arrhenius parameters [6]. The efficiency of reaction 7 is taken from Xu [13].

Num.	Reaction	k	A	β	Ea
1	$\mathrm{SOOT_nH + H \underset{k_{1b}}{\overset{k_{1f}}{\rightleftharpoons}} SOOT^*{}_n + H_2}$	k_{1f} k_{1b}	$1.000\ 10^{14}$ $1.439\ 10^{13}$	0 0	0 -37.63
2	$\mathrm{SOOT_nH + OH \underset{k_{2b}}{\overset{k_{2f}}{\rightleftharpoons}} SOOT^*{}_n + H_2O}$	k_{2f} k_{2b}	$1.630\ 10^{8}$ $1.101\ 10^{8}$	1.4 1.4	6.10 31.14
3	$\mathrm{SOOT^*{}_n + H \underset{k_{3b}}{\overset{k_{3f}}{\rightleftharpoons}} SOOT_nH}$	k_{3f} k_{3b}	$1.000\ 10^{13}$ 0	0 0	0 0
4	$\mathrm{SOOT^*{}_n + C_2H_2 \underset{k_{4b}}{\overset{k_{4f}}{\rightleftharpoons}} SOOT^*{}_nC_2H_2}$	k_{4f} k_{4b}	$3.500\ 10^{13}$ $3.225\ 10^{14}$	0 0	0 181.69
5	$\mathrm{SOOT^*{}_nC_2H_2 \underset{k_{5b}}{\overset{k_{5f}}{\rightleftharpoons}} SOOT_{n+2}H + H}$	k_{5f} k_{5b}	$1.000\ 10^{10}$ $8.770\ 10^{11}$	0 0	20.0 74.44
6	$\mathrm{SOOT^*{}_n + O_2 \overset{k_{6f}}{\rightarrow} SOOT^*{}_{n\text{-}2} + 2CO}$	k_{6f}	$1.000\ 10^{12}$	0	8.4
6'	$\mathrm{SOOT^*{}_nC_2H_2 + O_2 \overset{k_{6'f}}{\rightarrow} SOOT^*{}_n + 2HCO}$	$k_{6'f}$	$1.000\ 10^{12}$	0	6.4
7	$\mathrm{SOOT_nH + OH \overset{k_{7f}}{\rightarrow} SOOT^*{}_{k\text{-}2} + CH + HCO}$	k_{7f}	Efficiency $\gamma = 0.13$		

Table 1: HACA-RC mechanism

As shown by Rodrigues [11], applying a QSS assumption on the soot radicals SOOT^*_n, SOOT_nH & $\mathrm{SOOT}^*_nC_2H_2$ enables to write an approximation of both the surface growth and oxydation rates.

Surface growth corresponds to the addition of an ethylene molecule to a soot particle. Its rate $\dot{q}^{sg}$ is computed through :

$$\dot{q}^{sg} = q_4 \underset{QSS}{\simeq} \dot{q}_{4f}\chi_p - \dot{q}_{4b}\chi_c \tag{44}$$

Where - considering reaction i - q_i are reaction rates $[s^{-1}]$ and $\dot{q}_i$ are pseudo reaction rates that only accounts for the non-QSS species. χ factors are computed through the QSS (in [11] : Appendix 1)

$$\begin{cases} \chi_p & = \lambda \frac{\dot{q}_{2f}+\dot{q}_{1f}+\dot{q}_{3b}+\dot{q}_{7f}+\dot{q}_{5b}(1-\alpha)}{\dot{q}_{2b}+\dot{q}_{1b}+\dot{q}_{3f}+\dot{q}_{4f}\alpha} \\ \chi_c & = \lambda \frac{\dot{q}_{4f}}{\dot{q}_{4b}+\dot{q}_{6'f}+\dot{q}_{5f}} + \frac{\dot{q}_{5b}}{\dot{q}_{4b}+\dot{q}_{6f}+\dot{q}_{5f}} \\ \chi & = \lambda \end{cases}$$

With $\alpha = \frac{\dot{q}_{5f}}{\dot{q}_{4b}+\dot{q}_{6'f}+\dot{q}_{5f}}$ and $\lambda = 1/S_{C_2}$ a coefficient that takes active surface into account.

Oxidation corresponds to the loss of mass towards gas phase and is computed similarly :

$$\dot{q}^{ox} = q_6 + q_{6'} + q_7 \underset{QSS}{\simeq} \dot{q}_{6f}(\chi_p + \chi_c) + \chi \dot{q}_{7f} \tag{45}$$

The retroaction on the gas phase is taken into account for each species k through the pseudo reaction rates :

$$\text{sootConsumption}(k) = \sum_{k \in \text{reaction i}} \dot{q}_i \mathcal{A} \chi_x \tag{46}$$

Where the χ_x coefficient is either χ, χ_p or χ_c, depending on the reaction (the coefficient to be used comes from the QSS development).

The active sites concentration $\mathcal{A}$ $[mol.cm^{-3}]$ is computed as :

$$\begin{cases} \mathcal{A} & = \frac{\Delta q}{V_{C_2} N_A} \\ \Delta q & = \sum_{k=1}^{N_s} \frac{3}{\theta_k} V_{C_2}^{\frac{3-\theta_k}{3}} q(k) \left(V_{max,k}^{\frac{\theta_k}{3}} - V_{min,k}^{\frac{\theta_k}{3}} \right) \end{cases} \tag{47}$$

At this point, $\dot{q}^{sg}$ and $\dot{q}^{ox}$ gives the overall source term linked to surface chermistry. This source term must then be dispactched between each of the N_s sections. The source term of each section is once again splitted into three terms depending whether the mass comes from the gas phase, the section $k-1$ (surface growth) or $k+1$ (oxidation) ... For surface growth, the terms are computed as follow :

$$\begin{cases} \text{term}_k^1 & = \frac{1}{\rho} \frac{3}{\theta_k} \dot{q}^{sg} q_k V_{C_2}^{\frac{3-\theta_k}{3}} \left(\left(V_k^{max} - V_{C_2} \right)^{\frac{\theta_k}{3}} - V_{min,k}^{\frac{\theta_k}{3}} \right) \\ \text{term}_k^2 & = \frac{1}{\rho} \frac{3}{\theta_{k-1}} \dot{q}^{sg} q_{k-1} V_{C_2}^{\frac{3-\theta_{k-1}}{3}} \left(V_{max,k-1}^{\frac{\theta_{k-1}}{3}} - \left(V_{k-1}^{max} - V_{C_2} \right)^{\frac{\theta_{k-1}}{3}} \right) \\ \text{term}_k^3 & = \frac{1}{\rho} \frac{3}{\theta_{k-1}+3} \dot{q}^{sg} q_{k-1} V_{C_2}^{-\frac{\theta_{k-1}}{3}} \left(V_{max,k-1}^{\frac{\theta_{k-1}+3}{3}} - \left(V_{k-1}^{max} - V_{C_2} \right)^{\frac{\theta_{k-1}+3}{3}} \right) \end{cases} \tag{48}$$

The surface growth source term for each section is then computed as :

$$\dot{q}_k^{sg} = \underbrace{\text{term}_k^1}_{\text{gas} \to k} + \underbrace{\text{term}_k^2}_{\substack{k-1 \to k \\ \text{gas}}} + \underbrace{\text{term}_k^3}_{\substack{k-1 \to k \\ \text{soot}}} - \underbrace{\text{term}_{k+1}^3}_{k \to k+1} \tag{49}$$

For oxidation, the terms are computed as follow :

$$\begin{cases} \text{term}_k^1 & = \frac{3}{\theta_k} \frac{1}{\rho} \dot{q}^{ox} q(k) V_{C_2}^{\frac{3-\theta_k}{3}} \left(V_{max,k}^{\frac{\theta_k}{3}} - \left(V_k^{min} + V_{C_2} \right)^{\frac{\theta_k}{3}} \right) \\ \text{term}_k^2 & = \frac{3}{\theta(k+1)} \frac{1}{\rho} \dot{q}^{ox} q_{k+1} V_{C_2}^{\frac{3-\theta(k+1)}{3}} \left(\left(V_{k+1}^{min} + V_{C_2} \right)^{\frac{\theta(k+1)}{3}} - V_{min,k+1}^{\frac{\theta(k+1)}{3}} \right) \\ \text{term}_k^3 & = \frac{3}{\theta_k+3} \frac{1}{\rho} \dot{q}^{ox} q_{k+1} V_{C_2}^{-\frac{\theta(k+1)}{3}} \left(\left(V_{k+1}^{min} + V_{C_2} \right)^{\frac{\theta(k+1)+3}{3}} - V_{min,k+1}^{\frac{\theta(k+1)+3}{3}} \right) \end{cases} \tag{50}$$

And the source term for each section is computed accordingly :

$$\dot{q}_k^{ox} = -\underbrace{\text{term}_k^1}_{k \to \text{gas}} - \underbrace{\text{term}_k^2}_{\substack{k-1 \to k \\ \text{gas}}} + \underbrace{\text{term}_k^3}_{\substack{k-1 \to k \\ \text{soot}}} - \underbrace{\text{term}_{k-1}^3}_{k \to k-1} \tag{51}$$

Finally, the overall source term for section k can be computed as :

$$\dot{q}_k^s = \dot{q}_k^{nucl} + \dot{q}_k^{cond} + \dot{q}_k^{coag} + \dot{q}_k^{sg} + \dot{q}_k^{ox} \tag{52}$$

Source term is computed in $[cm^3.g^{-1}.s^{-1}]$. Before it is dumped into the discretized PBE (9), it is converted to SI units $[m^3.kg^{-1}.s^{-1}]$.

6 API

In this section, most of the soot-related methods are described.

6.1 Pre-treatment

BurnerFlame, ImpingingJet, CounterFlowDiffusionFlame, ... Soot computation is theoretically possible on any flame of type "axisymmetric stagnation flow". Activation of soot computation is made through the keyword "sections" when building the flame object.

```
f = ct.BurnerFlame(gas=gas, width=width, sections=sections)
f = ct.ImpingingJet(gas=gas, width=width, sections=sections)
...
```

soot_setup Sets sections informations to build them (precursors, collision model & trash section), soot chemistry (retroaction on gas phase, condensation of dimers, coagulation between particles, surface growth, particles oxidation & particles radiative heat loss) and whether section informations should be printed on screen or not.

```
f.soot_setup(precursors = list[str],
             retroaction =bool,
             condensation = bool,
             coagulation = bool,
             surface_growth = bool,
             oxidation = bool,
             fractal_aggregates = bool,
             radiation = bool,
             trash_section = float,
             show_sections = bool)
```

NB : Trash section is set by its diameter $[cm]$, any negative or null value will disable trash section.

This method calls several different properties, each corresponding to an aspect of soot chemistry. Eventhough it is not advised to use the properties outside of the *soot_setup* method, the advanced user will find these in Appendix A.

6.2 Treatment

restore + solve CANTERA's basic *restore* & *solve* methods. To be used on the first approach, might fail on tricky flames.

```
f.restore(filename = str, name = str)
f.solve(loglevel = int, refine_grid = str)
```

soot_source Return a dictionnary containing source terms ($[s^{-1}]$) :

- key = 'inception'
  (sections x points) array containing nucleation source terms,
- key = 'condensation'
  (sections x points) array containing condensation source terms,
- key = 'coagulation'
  (sections x points) array containing coagulation source terms,
- key = 'surface_growth'
  (sections x points) array containing surface growth source terms,
- key = 'oxidation'
  (sections x points) array containing oxidation source terms.

```
f.soot_source
```

Eventhough this method is more related to post-processing, it is introduced in this section as it must be used after solving the flame via *f.solve* (otherwise, source terms will be zeros).

soot_fluxes Returns a dictionnary containing soot fluxes ($[g.cm^{-3}.s^{-1}]$) :

- key = 'convection'
  (sections x points) array containing convective fluxes,
- key = 'diffusion'
  (sections x points) array containing diffusive fluxes,
- key = 'thermophoresis'
  (sections x points) array containing thermophoretic fluxes.

```
f.soot_fluxes
```

As same as *soot_source*, this method shall be used after solving the flame.

6.3 Post-treatment

soot_Y Returns a (sections x points) array containing the soot mass fraction for each grid point and each section.

```
f.soot_Y
```

soot_q Returns a (sections x points) array containing the volume distribtion of the soot volume $[cm^{-3}]$ at each grid point :

$$q_{soot,k} = \frac{\rho}{\rho_s} Y_{s,k} \frac{1}{V_k^{max} - V_k^{min}}$$

```
f.soot_q
```

soot_N Returns an (sections x points) array containing the soot particles number density $[cm^{-3}]$ in each section at each grid point :

$$N_{soot,k} = q_{soot,k} \log \frac{V_k^{max}}{V_k^{min}} = \frac{\rho}{\rho_s} Y_{s,k} \frac{\log \frac{V_k^{max}}{V_k^{min}}}{V_k^{max} - V_k^{min}}$$

```
f.soot_N
```

soot_fv Returns a (points) vector containing the soot volume fraction at each grid point :

$$f_V = \sum_{k=1+\text{first}}^{N_s+1-\text{last}} q_{soot,k} \left(V_k^{max} - V_k^{min}\right) = \frac{\rho}{\rho_s} \sum_{k=1+\text{first}}^{N_s+1-\text{last}} Y_{s,k}$$

Argmunents may be :

- *(first, last)* : indices of the first and last sections to be taken into account,
- *(min, max)* : Mean volumes $[cm^3]$ of the first and last sections to be tanken into account.

```
f.soot_fv(first = int, last = int)
f.soot_fv(min = float, max = float)
```

soot_Np Returns a (points) vector containing the soot particles number density $[cm^{-3}]$ at each grid point :

$$Np = \sum_{k=1+\text{first}}^{N_s+1+\text{last}} N_{soot,k} = \frac{\rho}{\rho_s} \sum_{k=1+\text{first}}^{N_s+1+\text{last}} Y_{s,k} \frac{\log \frac{V_k^{max}}{V_k^{min}}}{V_k^{max} - V_k^{min}}$$

Argmunents may be :

- *(first, last)* : indices of the first and last sections to be taken into account,
- *(min, max)* : Mean diameters $[cm]$ of the first and last sections to be tanken into account.

```
f.soot_Np(first = int, last = int)
f.soot_Np(min = float, max = float)
```

soot_psdf Returns a (sections) vector containing the soot particles size distribution $[cm^{-3}]$ at height HAB $[m]$ for each section.

Three types of PSDF can be plotted :

- if *out = 'Np'*, the PSDF will return :

$$PSDF = Np = \frac{\rho}{\rho_s} Y_s \frac{\log \frac{V^{max}}{V^{min}}}{V^{max} - V^{min}}$$

- if *out = 'Q'*, the PSDF will return :

$$PSDF = q = \frac{\rho}{\rho_s} Y_s \frac{1}{V^{max} - V^{min}}$$

- if *out = 'dNp/dlogd'*, the PSDF will return :

$$PSDF = \frac{1}{1 - \frac{\theta}{3}} q = \frac{1}{1 - \frac{\theta}{3}} \frac{\rho}{\rho_s} Y_s \frac{1}{V^{max} - V^{min}} \begin{cases} \theta = & 2 \text{ if } V_k^{mean} < v_{lim} \\ \theta = & 3 \log \frac{V_k^{mean}}{v_{lim}} + \frac{2}{3} \frac{\log \frac{V_{lim}}{V_{C_2}}}{\log} \frac{V_k^{mean}}{V_{C_2}} \text{ else} \end{cases}$$

  This is what is usually plotted versus the mobility diameter as $\frac{dNp}{d\log(dm)}$. However, the derivation of this quantity is unsure. What was done is related in Appendix D.

```
f.soot_psdf(HAB = float,
       out = str)
```

soot_dp Returns a (points) vector containing the primary particles diameter $[cm]$ at each point.

```
f.soot_dp
```

soot_np Returns a (points) vector containing the number of primary particles per aggregate at each point.

```
f.soot_np
```

flame.sections_min/max/mean_v Returns a (sections) vector containing the min/-max/mean volume of each section $[cm^3]$.

```
f.flame.sections_min_v
f.flame.sections_max_v
f.flame.sections_mean_v
```

flame.sections_mean/col_d Returns a (sections) vector containing the mean/collisional diameter of each section $[cm]$.

```
f.flame.sections_mean_d
f.flame.sections_col_d
```

flame.sections_mean_s Returns a (sections) vector containing the mean surface of each section $[cm^2]$.

```
f.flame.sections_mean_s
```

flame.soot_density Returns soot density $[g.cm^{-3}s]$

```
f.flame.soot_density
```

flame.sections_dp Returns a (sections) vector containing the primary particles diameter $[cm]$ of each section .

```
f.flame.sections_dp
```

flame.sections_np Returns a (sections) vector containing the number of primary particles per aggregate of each section.

```
f.flame.sections_np
```

7 Examples

Whatever the case, the sooting flame computation always looks the same :

1. Set-up the solution and the flame
2. Enable soot computation
3. Restore the non-sooting flame (computed with the same setup)
4. Solve the sooting flame

Two examples of sooting flames are introduced in the following.

7.1 Burner flame

The first flame is a laminar premixed flat-flame (ISF4 - premixed flame 2). It was experimentally studied by Xu et al. & Menon et al. . The flame is 4 cm long and produced by a 60mm McKenna burner. Three different equivalent ratios were measured. However, for this example, the focus will be set on the leanest flame (A). The scripting for both other flames would be the same, solely the gas composition should be changed. The charactertics of the flames are summed up in Tab. 7.1.

Flame	Equivalence Ratio Φ	$X_{C_2H_4}$	X_{O_2}	XN_2	V_{in}	T_{in}	P_{in}
A	2.34	0.1408	0.1805	0.6787			
B	2.64	0.1560	0.1773	0.6667	6.73 cm/s	298 K	1 atm
C	2.94	0.17	0.174	0.656			

Table 2: Burner flames

7.1.1 Computing the non-sooting flame

The computation of a sooting flame always starts by the computation of the non-sooting equivalent flame. Eventhough it is theoretically possible to compute a sooting flame from scratch, the computation will often crash (escpecially because of the retroation of the soot particles on the gas phase).

The computation of the non-sooting flame is performed the exact same way as in CANTERA 2.3-avbp :

- Import the Cantera library (and the numpy library that is used to load the experimental data)

```
import cantera as ct
import numpy as np
```

- Definition of the flame parameters

```
# Importing necessary libraries
import cantera as ct
import numpy as np

# Setting the mechanism name
mech = 'BIS'

#Setting pressure
pressure = ct.one_atm

# Setting temperature
temperature = 298.00

# Setting inlet velocity
velocity = 6.73e-2

# Setting flame initial composition
composition = 'C2H4:14.08, O2:18.05, N2:67.87'
```

- Definition of the gas object (through a *.cti* mechanism file) and its properties (T, P, $X_{species}$)

```
# Creating gas object
gas = ct.Solution('../%s.cti' % mech)
# Settng gas properties
gas.TPX = temperature, pressure, composition
```

- Definition of the flame object (specifying the gas and the domain's width) and its properties ($\dot{m}$). In our case, the temperature is imposed and the domain's width can thus be retrieved in the same file as the temperature :

```
# Loading experimental data (temperature vs. height)
z, T = np.genfromtxt('./T_vs_x.dat').T

# Creating flame object of the same size as the experimental flame
f = ct.BurnerFlame(gas=gas, width=z[-1]);

# Setting the flame's mass flow rate
f.burner.mdot = velocity * gas.density
```

- Setting up the computation parameters (energy and transport equations handling, refining criteria, ...)

```
# Disabling energy equation
f.energy_enabled = False

#Imposing proposed temperature profile
f.flame.set_fixed_temp_profile(z/max(z), T)
```

```
# Setting the transport model to mixture-averaged
f.transport_model = 'Mix'

# Setting mesh refinement criteria
f.set_refine_criteria(ratio=2.0, slope=0.1, curve=0.1)
```

- Running the computation and saving the non-sooting flame in a *.xml* file

```
# Solving the problem
f.solve(1, 'refine')

# Showing the result
f.show_solution()
s
# Saving the result
f.save('./RESULTS/%s.xml' % mech, 'non-sooting')
```

At this moment, a non-sooting flame solution is computed and stored.

7.1.2 Computing the sooting flame

Creating the flame object It is important to keep in mind that the sooting flame is a different object as the non-sooting flame. Indeed, the solution array is expanded by the number of sections : one cannot turn a non-sooting flame object into a sooting flame object. It is thue easoier to compute the sooting flame in a different script as the non-sooting flame. A benefit is that it is not necessary to compute the non-sooting flame each time one wants to modify a parameter on the sooting flame. The setup has to be the same as the one used to compute the non-sooting flame. Especially, *Solution* and *Flame* should be the same except for the number of sections. In order to keep computational cost reasonnably low, the number of sections will be set to 25.

```
# Importing necessary libraries
import cantera as ct
import numpy as np

# Setting the mechanism name
mech = 'BIS'

# Creating gas object
gas = ct.Solution('../%s.cti', % mech
)
# Loading experimental data (temperature vs. height)

z, T = np.genfromtxt('./T_vs_x.dat').T

# Creating flame object of the same size as the experimental flame with
↪  25 soot sections
```

```
f = ct.BurnerFlame(gas=gas, width=z[-1], sections=25)

# Setting transport model
f.transport_model = 'Mix'
```

Note that some computation parameters such as the inlet mass flow rate, temperature, pressure, gas composition, the energy equation handling and the refining criteria are stored in the *.xml* solution file and do not need to be defined again.

Setting up the soot computation At the moment, the solution array is ready to compute soot but no information on the sections were given and they are empty. In the next step, informations will be given to the code on how to build sections and how to compute soot source terms. Everything takes place is *soot_setup*.

In our case, Bisetti's mechanism (*BIS*) is used. This mechanism fits well with a mono-PAH approach involving Naphtalene (A2). The complete soot chemistry will be activated. No trash section will be used. *soot_setup* will thus be called as follow :

```
# Setting up soot calculation
f.soot_setup(precursors        =['A2'],
             retroaction       =True,
             condensation      =True,
             coagulation       =True,
             surface_growth    =True,
             oxidation         =True,
             fractal_aggregates=True,
             radiation         =True,
             trash_section     =-1)
```

Computation Once soot module is set-up, it is basically possible to solve the flame after restoring the non-sooting flame via :

```
# Restoring the non-sooting flame
f.restore('./RESULTS/%s.xml' % mech, 'non-sooting')

# Solving the sooting flame
f.solve(1,'disabled')
```

Two warning messages will be printed :

- one states that the number of species to be restored is smaller than the number of components of our flame. Indeed, when activating soot, the number of components was increased by $N_s = 25$,

- the other one concerns soot radiative heat transfers : we decided to compute soot radiative heat transfers while the energy equation is disabled. Soot radiative heat transfers won't thus be computed. In order to compute the soot radiative properties, energy equation must be activated.

Once the computation is over, the sooting flame can be displayed and saved.

```
# Displaying the result
f.show_solution()

# Saving the result
f.save('./RESULTS/%s.xml' % mech,'sooting')
```

The soot mass fraction is saved for each section at each point in the *sooting* simulation on the *.xml* solution file.

7.1.3 Post-treatment

In this tutorial, post-treatment takes place in a third script. It is strongly recommended to post-treat the flame in a standalone script in order not to compute the flames each time the post-treatment is executed.

Creating the flame object and activating the soot module As same as before, both the *Solution* and *Flame* objects must be creates and indentical as the previous ones.

Soot module will also be activated. As the geometrical informations on the sections are not store in the *.xml* file, they are recomputed. For the post-treatment to be accurate, they have to be the same as the ones the comptuation ran with : same precursors (especially the smallest and biggest ones), same trash section, same collision model. At the moment, CANTERA-soot is not able to retrieve the number of sections involved in a computation. Thus, the number of sections must also be the same.

```
# Importing necessary libraries
import cantera as ct
import numpy as np
import matplotlib.pyplot as plt

# Setting the mechanism name
mech = 'BIS'

# Creating gas object
gas = ct.Solution('../%s.cti' % mech
)
# Loading experimental data (temperature vs. height)
z, T = np.genfromtxt('./T_vs_x.dat').T
```

```
# Creating flame object of the same size as the experimental flame with
↪  25 soot sections
f = ct.BurnerFlame(gas=gas, width=z[-1], sections=25)

# Setting up soot computation
f.soot_setup(precursors        =['A2'],
             fractal_aggregates=True,
             trash_section     =-1
              )

# Restoring the sooting flame
f.restore('./RESULTS/'+mech+'.xml', 'sooting')
```

Note that it is not necessary anymore to give the transport model as equations won't be solved.

Generating post-treated data At this step, sections have been reconstructed and the flame object contains the previously computed soot mass fraction in each section. Python methods have been implemented reconstruct data of interest such as soot volume fraction, particle size distribution function, etc.

Data can now be plotted thanks to *matplotlib.pyplot* :

```
# Plot volume fraction
# Particles under 2nm and in last section are excluded
plt.figure(0)
plt.plot(f.grid * 100, f.soot_fv(min=2e-7, last=-1))
plt.yscale('log')
plt.ylabel('Volume fraction')
plt.xlabel('HAB ($cm$)')

# Plot particles number density
# Particles under 2nm and in last section are excluded
plt.figure(1)
plt.plot(f.grid * 100, f.soot_Np(min=2e-7, last=-1))
plt.yscale('log')
plt.ylabel('Soot particles number density ($cm^{-3}$)')
plt.xlabel('HAB ($cm$)')

# Displaying graphs
plt.show()
```

In this case, sections which mean diameter is smaller than 2 nm as well as the last section are excluded.

7.2 Impinging jet flame

The impinging jet flame corresponds to the ISF4 laminar premixed flame 6 and was studied by Rodrigues, Saggese, Li and Abid. This flame consists of a C_2H_4 / O_2-Ar flame at an equivalence ratio of 2.07, an inlet velocity of 8 cm/s and at atmospheric conditions. The flame perpendicularly impacts a flat plate which distance from the burner ranges from 40 mm to 2 cm.

7.2.1 Non-sooting and sooting flames

The preocedure is the very same as before except that the burner flame is replaced by an impinging jet. As there are several different flames to compute - which takes more time - the *.xml* files containig the sooting and non-sooting flames are given at :

```
./FILES/IMPINGING_JET/RESULTS/
```

The scripts used to compute these flames are given in Appendix B & C.

The flames were computed with the *KM2* mechanism. As shown by Blanquart [1], this mechanism fits well with pyrene, chrysene, benzo-a-pyrene, benzo-e-pyrene, perylene, benzo-perylene and coronene as precursors. A lumped-PAH approach will thus be used.

7.2.2 Post-treatment

Soot volume fraction and particle density As opposed to what was done in the burner flame, measurements are not taken alongside a single flame but at the end of the flame. Each width thus corresponds to a different flame. That's why post-treatment will be slightly different. Evethough this example does not aim at comparing with experimental data, the post-treatment will be executed as if.

First of all, the flame is set-up and the plotted variables are defined : z is the width of each flame, fv and Np will respectively be the soot volume fraction and particle number density at height z.

```
# Importing necessary libraries
import cantera as ct
import numpy as np
import matplotlib.pyplot as plt

# Setting mechanism name
mech='KM2'

#Storing flames names and size
widths = {'040':None, '045':None, '055':None, '060':None, '070':None,
              '080':None, '100':None, '120':None, '150':None,
              ↪  '200':None}
```

```
for k in widths:
    widths[k]=float(k) * 1.0e-4

# Creating the gas object
gas = ct.Solution('../%s.cti' % mech)

# Creating the vectors to be plotted
# Stagnation plate height
z = []
# Soot volume fraction
fv = []
# Soot particle number density
Np = []
```

Now, the script will loop over all the flames to append the vectors :

```
for width in widths:
    # Creating the impinging jet flame with 50 soot sections
    f = ct.ImpingingJet(gas=gas, width = widths[width], sections = 50)

    # Setting up soot computation
    f.soot_setup(precursors        =['A4', 'CHRYSEN', 'BAPYR',
     ↪  'BEPYREN', 'PERYLEN', 'BGHIPER', 'CORONEN'],
                 fractal_aggregates=True,
                 trash_section     =-1)

    # Restore sooting flame
    f.restore('./RESULTS/%s_%s.xml' % (mech, width), 'sooting')

    # Add the flame's size to the heights vector
    z.append(widths[width]*100)

    # Add the soot volume fraction at outlet to fv
    fv.append(f.soot_fv(min=2e-7, last=-1)[-1])
    # Add the soot particle density at outlet to Np
    Np.append(f.soot_Np(min=2e-7, last=-1)[-1])
```

At this point, z contains the length of each flame while *soot_fv* and *soot_Np* respectively contain the soot volume fraction and particle density number at the stagnation point. To plot the data, and compare it with experimental data and previous simulations :

```
#Plot the results
#fv
fix, (ax0, ax1) = plt.subplots(1, 2, sharex=True)
ax0.plot(z, fv, label='Cantera', color='g')
ax0.set_ylim((1e-10,1e-6))
ax0.set_yscale('log')
```

```
ax0.set_ylabel('$f_v$')
ax0.set_xlabel('HAB [$cm$]')
ax0.grid()

#Np
ax1.plot(z, Np, label='Cantera', color='g')
ax1.set_ylim((1e6,1e12))
ax1.set_yscale('log')
ax1.set_ylabel('$N_p$ [$cm^{-3}$]')
ax1.set_xlabel('HAB [$cm$]')
ax1.legend()
ax1.grid()

# Displaying graph
plt.show()
```

PSDF The soot Particles Size Distribution Function can be computed thanks to the *soot_psdf* method of the *FlowBase* class. The computed PSDF can thus be plotted for each flame :

- Same setup as usual

```
# Importing necessary libraries
import cantera as ct
import numpy as np
import matplotlib.pyplot as plt

# Setting mechanism name
mech='KM2'

#Storing flames names and size
widths = {'040':None, '045':None, '055':None, '060':None,
↪  '070':None,
              '080':None, '100':None, '120':None, '150':None,
              ↪  '200':None}
for k in widths:
    widths[k]=float(k) * 1.0e-4

# Creating the gas object
gas = ct.Solution('../%s.cti' % mech)

# Creating the plotting area
fig, ax = plt.subplots(2,5,sharex=True,sharey=True,figsize=(12,5))
ax[0,0].set_ylim((1e6,1e13))
ax[0,0].set_xlim((1e-7,1e-5))
ax[0,0].set_yscale('log')
ax[0,0].set_xscale('log')
i=0
```

```
j=0
k=1
```

- The PSDF is then plotted for each flame :

```
for width in widths:
    # Creating the impinging jet flame with 50 soot sections
    f = ct.ImpingingJet(gas=gas, width = widths[width], sections =
    ↪  50)

    # Setting up soot computation
    f.soot_setup(precursors        =['A4', 'CHRYSEN', 'BAPYR',
    ↪  'BEPYREN', 'PERYLEN', 'BGHIPER', 'CORONEN'],
                 fractal_aggregates=True,
                 trash_section     =-1)

    # Restore sooting flame
    f.restore('./RESULTS/%s_%s.xml' % (mech, width), 'sooting')

    # Plot the particle size distribution at each flame's outlet
    ax[i,j].plot(f.flame.sections_mean_d,
    ↪  f.soot_psdf(HAB=widths[width] type='dNp/dlogd'), color='g',
    ↪  label='Cantera')

    # Mark the flame's size as a title
    y = widths[width] * 100
    ax[i,j].set_title('H = %.2f $cm$' %  y)

    # Update next plot's position
    if (i>0):
        i = 0
        j += 1
    else:
        i+= 1
    k += 1
```

- Finally, the plotting area is finalized

```
# Finalize plotting area
ax0 = fig.add_subplot(111, frameon=False)
ax0.tick_params(labelcolor='none', top=False, bottom=False,
↪  left=False, right=False)
ax0.set_ylabel("$dN_p / d \log (d_m)$ [$cm^{-3}$]")
ax0.set_xlabel("d_m [$cm$]")

#Displaying graph
plt.show()
```

A Advanced methods

flame.soot_show_sections Displays section informations

```
f.flame.soot_show_sections
```

flame.soot_sections Gets the number of sections

```
f.flame.soot_sections
```

Setter is voluntarly not implemented to prevent memory allocation issues.

flame.soot_precursors Sets/gets the precursors, called by their name in the mechanism.

```
f.flame.soot_precursors = list[str]
f.flame.soot_precursors
```

After setting precursors, sections must be re-built via *f.flame.finalize_soot*.
Getter is not implemented at the moment.

flame.trash_section Sets/gets the minimal volume ($[cm^3]$) of the trash section.

```
f.flame.trash_section = float
f.flame.trash_section
```

After setting trash section, sections must be re-built via *f.flame.finalize_soot*

flame.soot_fractal_aggregates Sets/gets whether soot particles are fractal aggregates (*True*) or spheres (*False*).

```
f.flame.soot_fractal_aggregates = bool
f.flame.soot_fractal_aggregates
```

After setting aggregates fractality, sections must be re-built via *f.flame.finalize_soot*

flame.soot_do_retroaction Sets/gets whether retroaction on gas phase is computed or not.

```
f.flame.soot_do_retroaction = bool
f.flame.soot_do_retroaction
```

flame.soot_do_condensation Sets/gets whether condensation is computed or not.

```
f.flame.soot_do_condensation = bool
f.flame.soot_do_condensation
```

flame.soot_do_coagulation Sets/gets whether coagulation is computed or not.

```
f.flame.soot_do_coagulation = bool
f.flame.soot_do_coagulation
```

flame.soot_do_surface_growth Sets/gets whether surface growth is computed or not.

```
f.flame.soot_do_surface_growth = bool
f.flame.soot_do_surface_growth
```

flame.soot_do_oxidation Sets/gets whether oxidation is computed or not.

```
f.flame.soot_do_oxidation = bool
f.flame.soot_do_oxidation
```

flame.soot_do_radiation Sets/gets whether soot radiative heat transfers are computed or not.

```
f.flame.soot_do_radiation = bool
f.flame.soot_do_radiation
```

If energy is disabled, soot radiation won't be computed

B Non-sooting impinging jet flame

```
# Importing necessary libraries
import cantera as ct
import numpy as np

# Setting the mechanism name
mech = 'KM2'

# Setting temperature
T = 473.00

#Setting pressure
P = ct.one_atm

# Setting flame initial composition
X = 'C2H4:16.3, O2:23.7, Ar:60.0'

# Setting inlet velocity
v = 8.0e-2

# Storing each flame's name and width
widths={'040':None, '045':None, '055':None, '060':None, '070':None,
              '080':None, '100':None, '120':None, '150':None,
              ↪  '200':None}
for k in widths:
    widths[k]=float(k) * 1.0e-4

# Creating the gas object
gas = ct.Solution('../%s.cti' % mech)

for width in widths:

    # Setting the gas initial state
    gas.TPX = T, P , X

    # Loading experimental data
    zexp, Texp, Errexp = np.genfromtxt('./T_vs_z_'+width+'.dat').T

    # Creating the impinging jet flame
    f = ct.ImpingingJet(gas=gas, width = widths[width])

    # Setting the flame mass flow rate
    f.inlet.mdot = v * gas.density

    # Setting the stagnation plate's temperature
    f.surface.T = Texp[-1]
```

```
    # Setting transport model
    f.transport_model = 'Mix'

    # Setting mesh refinement criteria
    f.set_refine_criteria(ratio=2.0, slope=0.2, curve=0.2)

    # Setting initial flame
    f.set_initial_guess(products='equil')

    # Enablig radiative heat losses
    f.radiation_enabled = True

    # Solving the flame
    f.solve(loglevel=1, refine_grid='refine')

    # Displaying the solution
    f.show_solution()

    # Saving the flame
    f.save('./RESULTS/'+mech+'_'+width+'.xml','non-sooting')
```

C Sooting impinging jet flame

```
# Importing necessary libraries
import cantera as ct
import numpy as np

# Setting the mechanism name
mech = 'KM2'

# Setting temperature
T = 473.00

#Setting pressure
P = ct.one_atm

# Setting flame initial composition
X = 'C2H4:16.3, O2:23.7, Ar:60.0'

# Setting inlet velocity
v = 8.0e-2

# Storing each flame's name and width
widths={'040':None, '045':None, '055':None, '060':None, '070':None,
              '080':None, '100':None, '120':None, '150':None,
              ↪  '200':None}
for k in widths:
    widths[k]=float(k) * 1.0e-4

# Creating the gas object
gas = ct.Solution('../%s.cti' % mech)

for width in widths:
    # Creating the impinging jet flame with 50 soot sections
    f = ct.ImpingingJet(gas=gas, width = widths[width], sections=50)

    # Activating radiation
    f.radiation_enabled = True

    # Setting up soot compuation
    f.soot_setup(precursors        =['A4', 'CHRYSEN', 'BAPYR',
     ↪  'BEPYREN', 'PERYLEN', 'BGHIPER', 'CORONEN'],
                 retroaction       =True,
                 condensation      =True,
                 coagulation       =True,
                 surface_growth    =True,
                 oxidation         =True,
                 fractal_aggregates=True,
                 radiation         =True,
```

```
                trash_section     =-1)

    # Restoring the non-sooting flame
    f.restore('./RESULTS/%s_%s.xml' % (mech, width), 'non-sooting')

    # Solving the sooting flame
    f.solve(1,'disabled')

    # Displaying the solution
    f.show_solution()

    # Saving the sooting flame
    f.save('./RESULTS/%s_%s.xml' % (mech, width), 'sooting')
```

D About $\mathrm{d}Np/\mathrm{d}\log(d_m)$

Let the particle density number of section k : $N_{soot,k}$ $[cm^{-3}]$ be :

$$N_{soot,k} = q_k \log \frac{V_k^{max}}{V_k^{min}} \tag{53}$$

Where q_k is the volume density of volume fraction relative to section k and V_k are the maximal and minimal volumes of section k. Volumes might be written as :

$$V_k = \frac{1}{6 d_k S_k} \tag{54}$$

With d_k and S_k the corresponding diameter and section. Section is then written in therms of the fractal relation for soot :

$$\frac{S_k}{S_{C_2}} = \left(\frac{V_k}{V_{C_2}}\right)^{\frac{\theta}{3}} \tag{55}$$

Using both last equations, volume can be rewritten as :

$$V = \left(\frac{S_{C_2}}{6 V_{C_2}^{\frac{\theta}{3}}}\right)^{\frac{3}{3-\theta}} d^{\frac{3}{3-\theta}} \tag{56}$$

Rexpressing $N_{soot,k}$:

$$N_{soot,k} = \frac{3}{3-\theta} q_k \log\left(\frac{d_k^{max}}{d_k^{min}}\right) \tag{57}$$

$\mathrm{d}Np$ might then be interpreted as the increase in soot particle density thanks to each section, leading to :

$$\mathrm{d}Np_k = N_{soot,k} \tag{58}$$

Approximating the mobility diameter as the section mean diameter, $d\log(dm)_k$ gives :

$$\mathrm{d}\log(dm)_k = \log(d_k^{max}) - \log(d_k^{min}) = \log\left(\frac{d_k^{max}}{d_k^{min}}\right) \tag{59}$$

The discrete approximation of the derivative is straightforward :

$$\frac{\mathrm{d}Np}{\mathrm{d}\log(dm)}_k = \frac{3}{3-\theta} q_k \tag{60}$$

Using the definiton of q_k leads to the result.

Another approach consists in using the identity :

$$\frac{\mathrm{d}Np}{\mathrm{d}\log(dm)} = dm \frac{\mathrm{d}Np}{\mathrm{d}dm} \tag{61}$$

Which can be discretized at section k as :

$$\frac{\mathrm{d}Np}{\mathrm{d}\log(dm)}_k = d_k^{mean} \frac{N_{soot,k}}{d_k^{max} - d_k^{min}} \tag{62}$$

This formula give the same results as the previous one.

References

[1] G Blanquart and Pitsch H. *Combustion Generated Fine Carbonaceous Particles: Proceedings of an International Workshop held in Villa Orlandi, Anacapri, May*, chapter A joint volume-surface-hydrogen multi-variate model for soot formation. In [2], 2007.

[2] H Bockhorn, A D'Anna, A F Sarofim, and H Wang. *Combustion Generated Fine Carbonaceous Particles: Proceedings of an International Workshop held in Villa Orlandi, Anacapri, May.* 2007.

[3] CERFACS. Cerfacs chemistry website, 2020. [Online; accessed 25-February-2020].

[4] B. V. Derjaguin, A. I. Storozhilova, and Ya I. Rabinovich. Experimental verification of the theory of thermophoresis of aerosol particles. *Journal of Colloid And Interface Science*, 21(1):35–58, 1966.

[5] Paul S Epstein. On the Resistante Experienced by Spheres. (July):710–733, 1923.

[6] Fabian Mauss, Karl Netzell, and Harry Lehtiniemi. Aspects of modeling soot formation in turbulent diffusion flames. *Combustion Science and Technology*, 178(10-11):1871–1885, 2006.

[7] Fabian Mauss, Thomas Schäfer, and Henning Bockhorn. Inception and growth of soot particles in dependence on the surrounding gas phase. *Combustion and Flame*, 99(3-4):697–705, 1994.

[8] Michael F. Modest. *Radiative Heat Transfer (Third Edition)*. 2013.

[9] Michael Edward Mueller, Guillaume Blanquart, and Heinz Pitsch. A joint volume-surface model of soot aggregation with the method of moments. *Proceedings of the Combustion Institute*, 32 I(1):785–792, 2009.

[10] University of Adelaide. Isf 4 workshop, 2020. [Online; accessed 25-February-2020].

[11] Pedro Rodrigues. Modélisation multiphysique de flammes turbulentes suitées avec la prise en compte des transferts radiatifs et des transferts de chaleur pariétaux. 2018.

[12] Kermit C. Smyth and Christopher R. Shaddix. The Elusive History of m=1.157 - 0.56i for the Refractive Index of Soot. *Combustion and Flame*, 1(107):314–320, 1996.

[13] F. Xu, A. M. El-Leathy, C. H. Kim, and G. M. Faeth. Soot surface oxidation in hydrocarbon/air diffusion flames at atmospheric pressure. *Combustion and Flame*, 132(1-2):43–57, 2003.