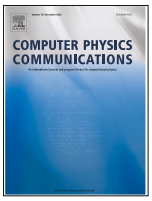
Computer Physics Communications 327 (2026) 110280

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc



Computer Programs in Physics

PPFM (Plasma Properties For Many): An object oriented C++ library for computing thermodynamic and transport properties of plasmas under different operating conditions

A. Vagnoni [a,b,*], M. Gherardi [a,b], E. Ghedini [a,b]

[a] Alma Mater Studiorum - University of Bologna, Bologna (BO), Italy
[b] Department of Industrial Engineering (DIN),

ARTICLE INFO

The review of this paper was arranged by Katharina Kormann

Keywords:
Plasma transport properties
Plasma thermodynamic properties
Plasma modeling
Collision integrals
Transport cross sections

ABSTRACT

Accurate modeling of thermal and non-thermal plasma systems requires reliable thermodynamic and transport properties, which are often difficult to measure or unavailable for many plasma mixtures and operating conditions. These quantities have a strong influence on the predictive quality of simulations. Current methods to compute these quantities for modeling may often lack the physical consistency required for high-fidelity applications, and computing them from very first principles can be cumbersome as they are the result of a complex problem that has to be systematically addressed. To overcome current challenges in computing plasma properties, we present PPFM (Plasma Properties For Many), an open-source C++ library for the computation of plasma thermodynamic and transport properties in both local and non-local thermodynamic equilibrium, LTE and NLTE, respectively. PPFM combines well-established theoretical models with a modular and advanced object-oriented architecture designed for flexibility, extensibility and scalability while ensuring minimal and intuitive user inputs. PPFM offers a promising reference platform to address thermodynamic and transport properties determinations, starting from very basic physical principles and microscopic properties, to compute macroscopic quantities.

PROGRAM SUMMARY
Program title: PPFM
CPC Library link to program files: https://doi.org/10.17632/2tdyjtch9y.1
Developer's respository link: https://doi.org/10.5281/zenodo.17191784
Licensing provisions: Creative Commons by 4.0 (CC by 4.0)
Programming language: C++
Nature of problem: Computing accurate thermodyanmic and transport properties for plasmas with simple input/output workflow in a commercial-like platform style.
Solution method: Building a modular and extensible library all around the physical model of thermodynamic and transport properties computations for plasma mixtures.

1. Introduction

Modeling has been crucial in unveiling the physical behaviors of complex plasma systems such as space weather, lightning, polar auroras, and fusion plasmas. Notably, computational fluid dynamics (CFD) simulations have improved the development and optimization of industrial thermal and non-thermal plasma devices, improving them in a wide range of applications. These include metal cutting, surface spraying, nano-powder synthesis, and more recently, bacteria inactivation in the medical and food chain sectors [1].

Simulations of gaseous flows under plasma conditions are complex tasks that require solving coupled mass, momentum, energy, and electromagnetic field equations. To gather useful information on mixtures, compounds, and plasma flow dynamics, these equations usually have to be solved in a multifluid and / or diffusion-driven approach, making the task even more difficult [2]. At this stage of modeling, accurate thermodynamic and transport coefficients of the plasma mixture must be provided, as they serve as known variables in the aforementioned governing equations and give great influence on final results. Thermodynamic properties can be derived from classical kinetic theory once the

* Corresponding author.
E-mail address: alberto.vagnoni3@unibo.it (A. Vagnoni).

https://doi.org/10.1016/j.cpc.2026.110280
Received 3 February 2026; Received in revised form 18 May 2026; Accepted 15 June 2026
Available online 22 June 2026

Nomenclature

T_e	Electron temperature [K]
T_h	Heavy particle temperature [K]
p	Pressure [Pa]
n_e	Electron particle density [#/m^3]
$n_{a^{(z)}}$	z-times ionized atom particle density [#/m^3]
n_m	Molecule particle density [#/m^3]
m_e	Electron mass [kg]
h	Planck constant [J·s]
Q^{int}	Internal partition function [#]
Q^{tr}	Translational partition function [#]
ϵ	Effective formation energy [J]
Z	Charge number [#]
$g_{s,l}$	lth Level degeneracy of sth specie [#]
$\epsilon_{s,l}$	lth Level energy of sth specie [J]
I_s	Ionization limit [J]
ΔI_s	Lowering ionization potential [J]
e	Electron charge [C]
ϵ_0	Vacuum dielectric constant [F/m]
k_B	Boltzmann Constant [J/K]
b	Collision parameter [Å]
r	Internuclear distance [Å]
$\phi(r)$	Interaction potential [eV]
E	Interaction energy of collision [eV]
χ	Deflection angle [°]
$Q^{(l)}$	Transport cross section [Å^2]
$\bar{\Omega}^{(l,s)}$	Adimensional collision integral [#]
μ	Reduced mass [kg]
γ	Reduced energy [#]
f	Particle distribution function [#]
ϕ	Distribution function perturbation [#]
$\vec{c}$	Particle velocity [m/s]
g	Relative velocity [m/s]
σ_{ij}	Differential scattering cross section [m^2/sr]
$d\Omega$	Solid angle element [#]

composition is known. The transport properties for partially ionized gases can be found by solving the Boltzmann equation. Unfortunately, correct computation of plasma thermodynamic and transport coefficients can be another complex task, especially for the latter, and can require significant human time and effort [1].

In the past few decades, the transport properties required for the closure of constitutive relations of the governing equations have been customarily calculated by analytical methods based on the first-order perturbation of the Boltzmann equation; i.e. the Chapman-Enskog expansion [3], which has been one of the most reliable methods to date. In order to use the Chapman-Enskog expansion to transport properties, the mass, momentum, and energy transfer resulting from colliding particles in the plasma mixture must be characterized through the calculation of collision integrals of the binary interactions within the mixture. Collision integrals are averages over Maxwellian distributions of the transport cross sections, which in turn can be interpreted as the geometrical area for the interacting particles to effectively exchange momentum during an interaction. Transport cross sections can also be computed by integrating the deflection angle of a binary encounter, which, in turn, can be calculated by integrating the interparticle potential of the force acting between that pair of particles [1].

Nevertheless, computation of transport properties remains a significant challenge as interaction potentials for colliding particles are not always available; then, researchers have to rely on different methods like numerical integrating experimental cross sections from databases or switching to quantum mechanical calculations from the phase shifts of the interacting particles. To complicate things even further, more correct properties arise when non-ideal phenomena like inelastic collisions, charge exchange between ions and parent atoms, and multi-state (excited) interacting particles are considered [4].

Thus, the solution of the Boltzmann equation through the modified Chapman–Enskog method is still largely neglected by the plasma community. Instead, practitioners often rely on precomputed values for specific gas mixtures and operating conditions — obtained either by interpolating tabulated data, fitting polynomial expressions as functions of temperature and pressure, or applying approximate mixing rules based on pure gas properties [5] — in order to reduce computational cost or compensate for the lack of detailed collision data.

Several commercial and academic software packages, such as *CEA*, *EGLIB*, and *CHEMKIN*, can provide useful properties for plasma CFD computations, but are limited when it comes to highly ionized thermal and chemical non-equilibrium plasmas. To the authors' knowledge, only the *MUTATION++* [6] library offers accurate transport coefficients under the non-local thermodynamic equilibrium (NLTE) assumption. However, its reliance on interpolating hard-coded databases restricts its flexibility and scope since these databases are limited to the detailed studies available in the literature. Another notable tool is the *minplascalc* [7,8] Python software tool developed by MINTEK; released in 2024, the minplascalc tool had the same objective as this work: offering an open source tool for thermodynamic and transport properties calculations of plasma mixtures. However, the *minplascalc* tool is dedicated only to LTE plasmas, its design remains much less modular than the one described in this work, and relies on empirical formulas for interaction characterizations instead of offering ab initio transport cross sections and collision integrals calculations. To address these limitations, a more general and adaptable tool is needed, capable of computing thermodynamic and transport properties and their underlying data from intensive physical information and, when necessary, from first principles. Such a tool should also be effectively suited to enlarge the existing databases with newer plasma mixtures and different conditions, while minimizing researchers' efforts.

For this purpose, the authors developed PPFM, an Object-Oriented C++ library designed to simplify and centralize all the useful routines researchers use for determining plasma overall properties numerically, gathering those routines together, and offering precise algorithms for ab initio calculations. In doing this, a strong effort has been dedicated to make advantage of C++ Object Oriented design. Objectification of physical and mathematical entities have been stressed in order to build up a strongly modular structure that will be easily extensible and easy to use with very few rules and minimal user input. In the next sections, this structure will be explained, along with some functionalities and implementation techniques. Next, an illustrative example with the reference mixture of ArH_2 will be shown in the comparison and validation section to demonstrate and highlight the ease of use and versatility of this tool, and then the results will be used to verify the accuracy of the PPFM calculated values with previous studies by our research group.

PPFM has been developed re-engineering the legacy C code used for the data published in [9,10] and [11] articles into the object-oriented framework described in the previous paragraph. The original codes were organized as a collection of C functions and scripts and were originally developed as internal research software of our research group at the University of Bologna. Although not publicly distributed at the time, it constituted the computational backbone for the present PPFM implementation.

1.1. Software overview

The main goal in the development of PPFM is to provide a platform that facilitates access to thermodynamic and transport properties calculations (TTP in the following sections), thus reducing the knowledge barrier required for the customization or the usage of these kinds of calculations. By analyzing the scientific literature within this problem [9–19] it is possible to identify a common workflow for determining the TTP of partially to fully ionized plasma mixtures with the kinetic theory

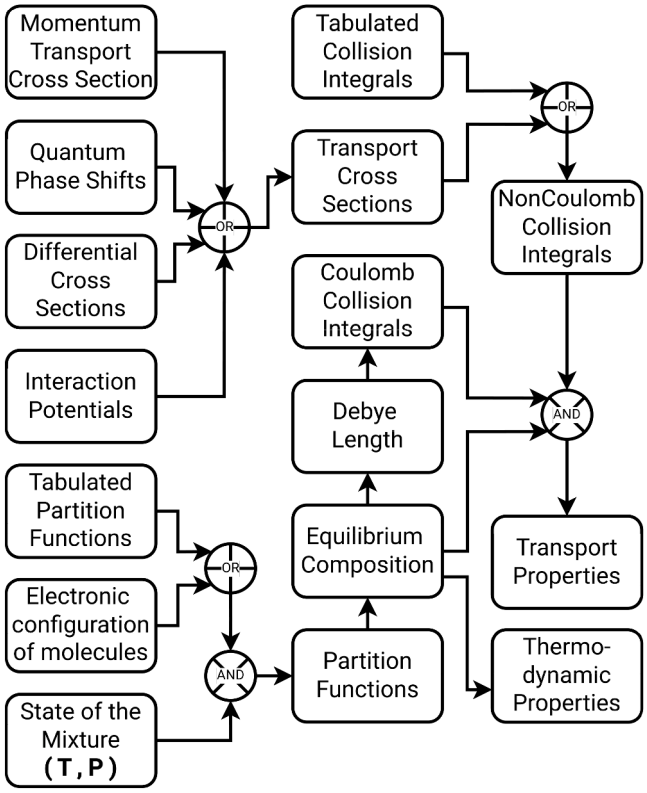


Fig. 1. Workflow for the determination of thermodynamic and transport properties of plasma gas mixtures.

of gases and the modified Chapman-Enskog solution of the Boltzmann equation.

Such workflow is depicted in Fig. 1, PPFM is build on this workflow and its users can use the capabilities of the software in order to stream the workflow bottom-to-top; from data sources (left part of the scheme) through a middle stage of computation of, mainly, the collision integrals and Partition Functions, to the desired TTP. Once the Partition Functions for the chemical species are known, the equilibrium composition can be computed and suddenly the thermodynamic properties of the mixture through finite-difference formulas. The transport properties are more complex as they require the composition of the plasma and the characterization of the whole ensemble of distinct binary interactions within the mixture through the computation of collision integrals.

The workflow also shows the possibility for a PPFM user to choose from different data sources for transport cross sections determinations: mandatory, in many cases, for the lack of reference data. To address a TTP calculation, a researcher must define which chemical species have to be considered within the mixture along with its state (temperature and pressure). The chemical species to be considered in a plasma can be numerous, and all of them must come at least with mass, charge number, and their formation and ionization energies. Then, N being the number of chemical species within the mixture, a researcher has to compute (or collect) N partition functions and $N(N+1)/2$ collision integrals, one for each binary interaction.

These two data sets are sufficient for the chemical composition and the Chapman-Enskog solution of the Boltzmann equation; unfortunately, their computation itself is indeed a challenging task. Molecular partition functions, for example, have to deal with the vibrational and rotational states of molecules, whereas a threefold integration, usually with singularities, is required to compute collision integrals from the interaction potential into the deflection angle, the transport cross section, and the former collision integral. Then, the computed data have to be systematically collected and handled for the following computations.

In order to overcome this and more of the complexities within the problem of determining TTP, our program has been developed taking advantage of modern C++ paradigms like inheritance and template meta-programming in order to automate the calculations and other complex behaviors typical of this problem whenever possible.

The use of these paradigms will be described in the following sections after a brief overview over the models that PPFM currently implements.

2. Theoretical background

Although an extensive examination of TTP determination is outside the scope of this paper, a synopsis of the PPFM implemented formulas is needed. The nomenclature for this section can be found on the first page of this paper.

2.1. Composition and thermodynamics

Composition, i.e. particle densities of chemical species, is required for thermodynamic and transport properties in both LTE and NLTE plasmas. In two-temperature plasmas, particle densities can be expressed as functions of electron temperature and the non-equilibrium parameter $\theta = T_e/T_h$. To compute composition, our software currently employs the equations of Van de Sanden et al. modified for the two-temperature approach [20].

$$\frac{n_{\mathrm{e}} n_{\mathrm{a(z+1)}}}{n_{\mathrm{a(z)}}} = 2\left(\frac{2\pi m_{\mathrm{e}} k_B T_{\mathrm{e}}}{h^2}\right)^{3/2} \frac{Q^{\mathrm{int}}_{\mathrm{a(z+1)}}(T_{\mathrm{e}})}{Q^{\mathrm{int}}_{\mathrm{a(z)}}(T_{\mathrm{e}})} \exp\left(-\frac{\epsilon_{\mathrm{a(z+1)}}}{k_B T_{\mathrm{e}}}\right), \tag{1}$$

$$\frac{n_{\mathrm{a}}^2}{n_{\mathrm{m}}} = \frac{(Q^{\mathrm{tr}}_{\mathrm{a}}(T_{\mathrm{h}}))^2}{Q^{\mathrm{tr}}_{\mathrm{m}}(T_{\mathrm{h}})} \frac{(Q^{\mathrm{int}}_{\mathrm{a}}(T_{\mathrm{e}}))^2}{(Q^{\mathrm{int}}_{\mathrm{m}}(T_{\mathrm{h}})} \exp\left(-\frac{\epsilon_{\mathrm{m}}}{k_B T_{\mathrm{h}}}\right), \tag{2}$$

$$\sum_i^{\mathrm{species}} n_i Z_i = 0\,, \tag{3}$$

$$p = n_{\mathrm{e}} k_B T_{\mathrm{e}} + \sum_i^{\mathrm{heavy}} n_i k_B T_{\mathrm{h}}. \tag{4}$$

Where the formation energy for electrons and neutral atoms is null, for positive ions it is equal to the summation over the previous states ionization energies; for molecules and for anions it is the bond dissociation energy and the electron affinity, respectively, both with negative sign.

This equation set is solved via a matrix-based algorithm proposed by Godin and Trépanier in [21], which had been generalized for mixtures of M elements, while prior legacy C code versions of our software were limited to binary mixtures.

The partition functions needed to Eqs. (1)–(4) can be hard coded with temperature-dependent polynomials, interpolated from data tables in input files, or computed from electronic configurations, with respect to atomic partition functions, using the simple formula

$$Q^{\mathrm{int}}_{\mathrm{a}}(T) = Q^{\mathrm{el}}_{\mathrm{a}}(T) = \sum_{l=0}^{\epsilon_{max}=I_s-\Delta I_s} g_{s,l} \exp\left(-\frac{\epsilon_{s,l}}{T}\right) \tag{5}$$

which have to be cut under the ionization limit with the Debye-Huckel cut-off criterion,

$$\Delta I_s = \frac{e^2(Z+1)}{4\pi\varepsilon_0 \lambda_D}\,, \tag{6}$$

λ_D being the Debye length.

Thermodynamic functions such as specific heat under constant pressure can be computed from the chemical composition with little effort using the classical kinetic formulation and finite differences

$$c_p = \left(\frac{\partial e}{\partial T}\right)_P + R'\left[1 + \frac{T}{R'}\left(\frac{\partial R'}{\partial T}\right)_P\right] \tag{7}$$

where e is the internal energy and $R' = P/\rho T$ is the specific gas constant. Then the mass density is

$$\rho = \sum_{i=1}^{N} n_i m_i \tag{8}$$

while for the total specific enthalpy

$$H = \frac{5}{2}\frac{k}{\rho}\sum_i n_i T_i + \frac{1}{\rho}\sum_i n_i \epsilon_i + \frac{k}{\rho}\sum_i n_i T_i^2 \frac{\partial \ln Q_i^{\text{int}}}{\partial T_i}. \tag{9}$$

2.2. Collision integrals and transport properties

In PPFM, collision integrals can be interpolated from raw data files or can be calculated in different ways. When using interaction potentials, the program evaluates the deflection angle χ, the transport cross section $Q^{(l)}$, and finally the dimensionless collision integral $\bar{\Omega}^{(l,s)}$ through numerical integration.

$$\chi(b,E) = 2b\int_{r_m}^{\infty}\left[1 - \frac{b^2}{r^2} - \frac{\varphi(r)}{E}\right]^{-1/2}\frac{dr}{r^2} \tag{10}$$

is the deflection angle while

$$Q^{(l)}(E) = 2\pi\int_0^{\infty} b\left(1-(-1)^l\cos^l\chi\right)db \tag{11}$$

is the transport cross section. An alternative to this

$$Q^{(l)}(g) = 2\pi\int_0^{\pi}\sigma(g,\chi)\,(1-\cos\chi)\,\sin\chi\,d\chi \tag{12}$$

can be used when the differential cross section $\sigma(g,\chi)$ is known, for example, from experimental data or database resources. Finally, the dimensionless collision integral

$$\bar{\Omega}_{ij}^{(l,s)}(T_{ij}^*) = \frac{2(l+1)}{(s+1)!\left(2l+1-(-1)^l\right)}\int_0^{+\infty} e^{-x}x^{s+1}Q_{ij}^{(l)}(x)\,dx \tag{13}$$

with $x = \mu_{ij}g^2/2k_B T_{ij}^*$ and the effective temperature of collision

$$T_{ij}^* = \left(\frac{1}{m_i+m_j}\left(\frac{m_i}{T_j}+\frac{m_j}{T_i}\right)\right)^{-1} \tag{14}$$

PPFM implements two algorithms to handle the singularity in the integral of the deflection angle: an adaptive method [22] for potentials with a clear turning point and a faster averaging method [23] for monotonic interactions. The choice is resolved at compile time via template `const expression` evaluations.

If quantum effects are relevant (e.g., resonant or very light-particle collisions), transport cross sections can be computed from quantum phase shifts of the interacting particles [24]

$$Q^{(1)} = \frac{4\pi}{\kappa^2}\sum_{l=0}^{\infty}(l+1)\sin^2(\delta_l - \delta_{l+1}) \tag{15}$$

$$Q^{(2)} = \frac{4\pi}{\kappa^2}\sum_{l=0}^{\infty}\frac{(l+1)(l+2)}{2l+3}\sin^2(\delta_l - \delta_{l+2}) \tag{16}$$

$$Q^{(3)} = \frac{4\pi}{\kappa^2}\sum_{l=0}^{\infty}\left(\frac{l+1}{2l+5}\right)\left\{\frac{(l+3)(l+2)}{2l+3}\sin^2(\delta_l-\delta_{l+3}) + \frac{2(l^2+2l-1)}{2l-1}\sin^2(\delta_l-\delta_{l+1})\right\} \tag{17}$$

$$Q^{(4)} = \frac{4\pi}{\kappa^2}\sum_{l=0}^{\infty}\frac{(l+2)(l+1)}{(2l+7)(2l+3)}\left\{\frac{(l+4)(l+3)}{2l+5}\sin^2(\delta_l-\delta_{l+4}) + \frac{2(2l^2+6l-3)}{2l-1}\sin^2(\delta_l-\delta_{l+1})\right\} \tag{18}$$

where κ is the wave number and δ_l are the computed phase shifts. For charge exchange, a simple empirical form is used for the transport cross section

$$Q^{(l)} = (A - B\ln g)^2 \tag{19}$$

or its analytical integration into the collision integral [24], as

$$\bar{\Omega}^{(l,s)} = A^2 - ABx + \left(\frac{Bx}{2}\right)^2 + \frac{B\zeta}{2}(Bx-2A) + \frac{B^2}{4}\left(\frac{\pi^2}{6} - \sum_{n=1}^{s+1}\frac{1}{n^2} + \zeta^2\right) + \frac{B}{2}[B(x+\zeta)-2A]\ln\frac{T_{ij}^*}{M} + \left(\frac{B}{2}\ln\frac{T_{ij}^*}{M}\right)^2 \tag{20}$$

where A and B are experimental constants, $x = \log(4R)$, $\zeta = \sum_{n=1}^{s+1}\frac{1}{n} - \gamma$ and γ is the Euler constant.

If inelastic collisions are to be accounted , the total collision integral becomes

$$\bar{\Omega}^{(l,s)} = \sqrt{\left(\bar{\Omega}_{\text{el}}^{(l,s)}\right)^2 + \left(\bar{\Omega}_{\text{in}}^{(l,s)}\right)^2} \tag{21}$$

In the end, for systems with multiple interaction states (e.g. metastable and excited states), PPFM computes

$$\bar{\Omega}^{(l,s)} = \frac{\sum_k g_k\bar{\Omega}_k^{(l,s)}}{\sum_k g_k} \tag{22}$$

where g_k are the degeneracies of the considered states.

Finally, the computation of transport properties is done using the Chapman-Enskog method to solve the Boltzmann equation, assuming that the particle distribution function is a first-order perturbation to the Maxwellian distribution.

$$\frac{Df_i}{Dt} = \sum_{j=1}^{N}\iint(f_i'f_j' - f_if_j)g\sigma_{ij}d\Omega d\vec{c}_j\,, \tag{23}$$

$$f_i = f_i^0(1+\phi_i)\,, \tag{24}$$

$$\frac{Df_1^{(0)}}{Dt} = \iint f_1^{(0)}f^{(0)}(\phi_1' + \phi' - \phi_1 - \phi)g\sigma_{11}d\Omega d\vec{c} + \sum_{j=2}^{N}\iint f_1^{(0)}f_j^{(0)}(\phi_1' - \phi_1)g\sigma_{1j}d\Omega d\vec{c}_j \tag{25}$$

$$\frac{Df_i^{(0)}}{Dt} = \iint f_i^{(0)}f_1^{(0)}(\phi_1' - \phi_1)g\sigma_{i1}d\Omega d\vec{c}_1 + \sum_{j=2}^{N}\iint f_i^{(0)}f_j^{(0)}(\phi_i' + \phi_j' - \phi_i - \phi_j)g\sigma_{ij}d\Omega d\vec{c}_j \tag{26}$$

Then, all transport coefficients resulting from this method will depend on matrix determinants whose elements are a linear combination of collision integrals and the plasma composition, commonly referred to as *bracket integrals*. These *bracket integrals* can be found by expanding the solutions to integral Eqs. (25) and (26) in some orthogonal polynomials called Sonine polynomials.

The explanation of the method far exceeds the scope of this paper; the interested reader can refer again to [3] and to [25] for further details.

At the moment, the code implements two developments of the Chapman-Enskog approximation for transport properties: the ones due to R.S.Devoto [26,27] and those from X.N.Zhang, A.B.Murphy et al. [28], the latter being limited for now to classical transport coefficients like electrons and heavy particles translational thermal conductivities, viscosity, and electrical conductivity. The second theory is different from the first for the retention of the first term of Eq. (26), and for the definition of diffusion driving forces ($\vec{d}_j$); first adopted by Rat et al. [29] which leads to more correct formulation of the ordinary diffusion coefficients. Both implemented theories allow for the calculations of the transport coefficients up to the 4th order of the expansion. The validation for the values resulting from Devoto decoupled theory is demonstrated through comparison with computed values for the Argon Hydrogen mixture [11], while the Zhang values are confronted with the data calculation proposed in [30] for a pure Argon plasma considering four chemical species Ar, Ar^+, Ar^{+2} and e^-.

Explicit formulas for transport properties (e.g., viscosity, thermal conductivities, and diffusion coefficients) are not reported here because they are thoroughly detailed in the cited studies. Their inclusion would

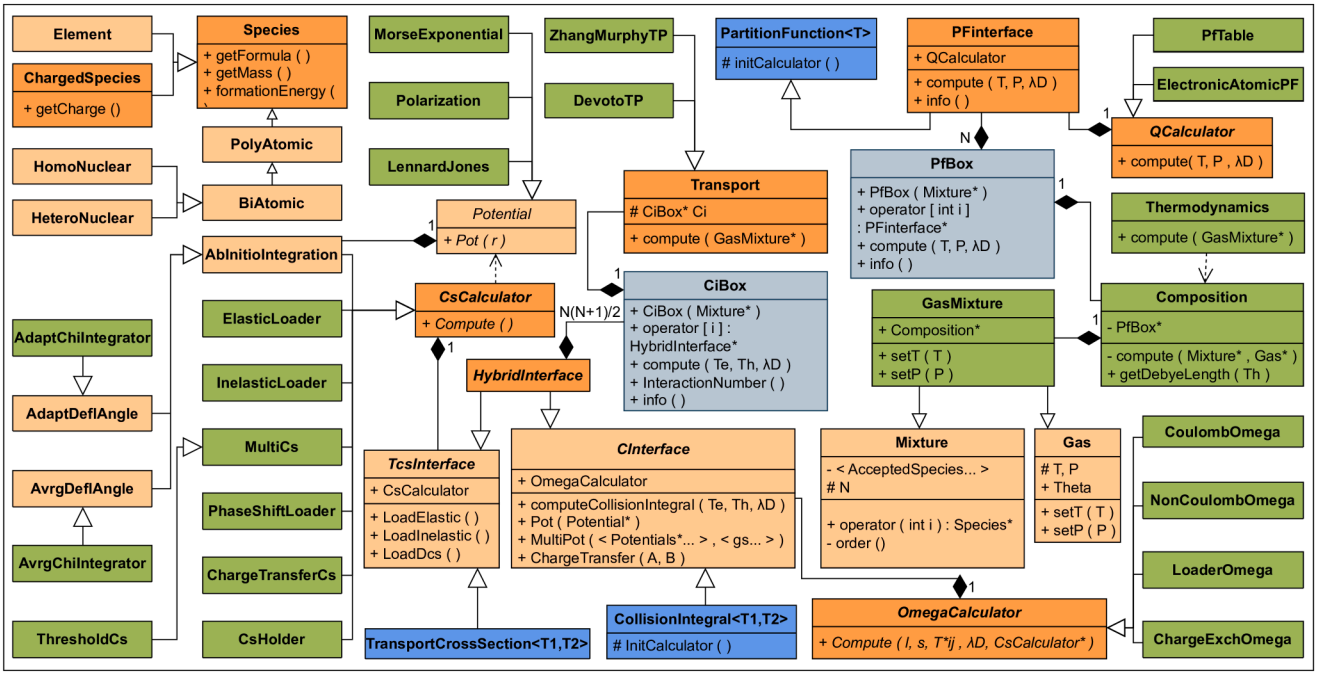


Fig. 2. Unified Modeling Language (UML) Class Diagram for PPFM class hierarchies design. Empty triangles arrows are for inheritance relations, black diamond for composition and dashed lines for dependencies. For reasons of simplicity the most of the methods and members of classes have not been reported. The complete UML class diagram is available from the authors upon request.

be redundant and unnecessarily verbose, given that the focus of this section has been on the collection and computation of the input data required for TTP evaluation, which is, in fact, the most challenging task related to plasma TTP determination within this approach.

3. Architecture

The development of PPFM has been carried out closely following the Single Responsibility Principle (SRP)[1] of object-oriented programming. When applied to scientific codes, these usually result in a class structure in which each physical or mathematical entity is encapsulated in a distinct class. For PPFM, one-to-one correspondence between mathematical and physical entities and program classes has been attempted whenever this could be useful.

This design significantly improves modularity, extensibility, and debugging. In PPFM, this paradigm has proven to be essential for managing the complexities of TTP determination. In detail, it enhances the clear separation between physical concepts and their underlying computational logic, while offering a user-facing interface that is both powerful and intuitive. These goals are achieved by relying on deep class hierarchies and extensive use of template meta-programming techniques.

3.1. Modularity

In accordance with the SRP, PPFM counts more than sixty implemented classes, excluding chemical species that are also implemented as standalone classes. All implemented classes can be logically grouped into eight macro-modules, resulting in deep class hierarchies with clearly separated interfaces.

- **Chemical Species:** the internal database of the program. Each species is implemented as a concrete class, while abstract classes define conceptual categories such as elements, ions, and molecules.
- **Potentials:** classes that model interparticle forces through potential energy functions. Each class provides a formula for the potential as a function of the interparticle distance r, used in the integration of deflection angles.
- **Cross Sections:** classes dedicated to the calculation and handling of transport cross sections between interacting species from different data sources and methods.
- **Collision Integrals:** classes responsible for the computation and management of interaction collision integrals, which depend on transport cross sections.
- **GasMixture:** the main entry point for initializing calculations. It contains state variables and the list of chemical species and a state-dependent composition object.
- **Partition Functions:** classes that compute and store partition functions for each species, used in composition and thermodynamic properties calculations.
- **Composition and Thermodynamics:** classes to compute and store species number densities and thermodynamic properties, based on the state of the mixture and its related partition functions.
- **Transport Properties:** classes that compute and store transport properties. These depend on both the mixture (and its composition) and the collision integrals.

The Unified Modeling Language (UML) class diagram reported in Fig. 2 shows the most important classes of the program and their coded relationships.

The diagram shows concrete classes, interface classes that can be used by the user, totally abstract classes used in the code for the semantic separation of entities, and for members & methods encapsulation, template classes, and box classes; in green, orange, light orange, blue, and gray, respectively. Users can directly initialize and operate on concrete and box classes. Interface classes can be used as references to polimorphically initialize and operate concrete classes in the program environment or in new implementations through the `new` C++ operator. The template classes are initialized by the box classes to dispatch algorithms and behaviors at compile time.

[1] *"Gather together things that change for the same reasons, separate those that change for different reasons." — R.C. Martin* [31]

```
Error in the 0−th collision integral: Ar_Ar
    Pointer "phi" for Ar_Ar is null.

Initialize in main:
  ciboxname[0]−>setPot( see Potential.h for more );

otherwise:
− try loading it from Raw files placed in:
    data/CollisionIntegral
− prefer a different calculation method,
  see CollisionIntegral.h for more.
```

Listing 1. Example of runtime error message generated by PPFM for an invalid collision integral definition.

```
template<typename T1 ,typename T2 >
void CollisionIntegral<T1,
    ↪ T2>::InitCalculator(bool) {

    if constexpr (
        ↪ std::is_base_of_v<ChargedSpecies,
        ↪ T1> &&
        ↪ std::is_base_of_v<ChargedSpecies,
        ↪ T2> )
        Omega = new CoulombOmega(new T1, new T2);
    else
        Omega = new NonCoulombOmega(this);
}
```

Listing 2. Implementation of the `OmegaCalculator*` assignement during a `CollisionIntegral<T1,T2>` object construction.

This architecture gives the user a high degree of flexibility in defining the problem itself and in selecting computational methods and models. Such flexibility is made possible through the hybrid use of run-time and compile-time polymorphisms, with a preference on compile-time to reduce run-time overhead while preserving flexibility.

The modular design of PPFM has been critical for development and testing, facilitating debugging, and ensuring extensibility. The chemical species framework, for example, allows developers to model a wide range of species, organized into clearly defined categories.

To extend the species database, users need only to derive a concrete class from one of the existing interface classes. For instance, `MolecularHydrogenI` (H_2^+) will inherit from both `ChargedSpecies` and `HomoNuclearBiatomicMolecules`, the latter being a specialization of a more generic `PolyAtomicMolecule`, and, ultimately, every chemical species class parenthood results in them being all derived from the `Species` abstract class.

A very similar approach is retained for algorithm characterization and implementation; that is, each method to compute the collision integrals and the transport cross sections will be derivations of the `OmegaCalculator` and the `TcsCalculator` abstract interface classes, respectively.

PPFM also follows best practices for encapsulation by declaring most internal data and methods as `private (-)` or `protected (#)`, exposing to users only what is strictly necessary through public interfaces. This enhances modularity, but also improves security and interoperability.

3.2. Container classes

Encapsulation, combined with interface-driven design, allows for the implementation of container classes that manage complex collections of physical entities. As shown in Figs. 1 and 2, and through everything described in the theoretical background section, the most natural candidates for containerization are the N partition functions and $N(N+1)/2$ collision integrals.

To this end, PPFM implements the `PfBox` and `CiBox` classes, which safely store pointers to `PfInterface` and `HybridInterface`, respectively. These interfaces allow the dynamic instantiation and operation of concrete classes such as `PartitionFunction<T>` and `CollisionIntegrals<U1,U2>`, where the template parameters denote the actual types involved, without losing crucial information through the species hierarchy.

For clarity and consistency, all species-related collections are alphabetically ordered from the `Mixture` class definition, relying on a fixed IUPAC-like chemical nomenclature provided by the `formula()` method common to all species classes.

A minimal yet useful `info()` method is implemented for the user to inspect stored objects. Additionally, `Box` classes implement the `[i]` operator to access member functions through their interface types, in order to help the user follow the workflow. If something goes wrong anyway, exceptions will be cast with suggestions for the user, and execution will stop, see code snippet (1). The same strategy of exception casting is retained and attempted in code bottlenecks where calculations may fail to converge, so that users and developers can identify and correct implementations with common debug resources.

The operability of the objects through the environment of the program, which is, for now, the implementation of the `main.cpp` file for its execution, is demonstrated in the source code reported in the appendix code (3) for the Argon - Hydrogen validation.

3.3. The template - based approach

Beyond offering access to the user-facing interface, PPFM must also automate much of the internal behavior not explicitly defined by the user. To enable this, the container classes instantiate the template-based concrete types `PartitionFunction<T>` and `CollisionIntegrals<T1,T2>`. Then, its the template classes that will do the most of the work not specified by the user.

This use of *generic programming* [31] allows PPFM to apply compile-time polymorphism by dispatching types and algorithms through C++ concepts and constraints. To illustrate this mechanism, consider the `CollisionIntegral<T1,T2>` template.

The constraint: `if constexpr` depicted in the code snippet (2) is evaluated at compile time to ensure that the appropriate `OmegaCalculator` instance, either `CoulombOmega` or `NonCoulombOmega`, is used based on the types of colliding species. This behavior is repeated when selecting an `AvrgChiIntegrator` or `AdaptChiIntegrator` whenever the model potential used to model an interaction is monotonic or not. Moreover, and more important indeed, is that in this way the information of a chemical species being a particular one, for example a `charged species` or a molecule, is retained at compile time, without having to resolve it every time at run-time through redundant `dynamic_casts` constructs.

This strategy of validating type properties at compile time reduces the need for run-time type resolution, which can be computationally more demanding in compiled languages. However, the overall performance of the framework depends on broader architectural trade-offs, as discussed in the following section.

These abstract concepts [31] are implicitly enforced throughout the species hierarchy implementation depicted in Fig. 2, where all the chemical species are `Species`, but some are `Elements`, others are `Charged Species`, `PolyAtomicMolecules` and so on.

The template-based architecture contributes significantly to the extensibility of PPFM. To support new models or approximations, users can define custom types that inherit from `OmegaCalculator` or `ParfCalculator` and implement a new `compute()` method. Then he has to implement the logic for the compile-time choosing of the appropriate algorithm during the construction of the template classes, otherwise, it

Table 1
Comparison of plasma-property tools based on workflow and modeling approach.

Feature	PPFM	MUTATION++	minplascalc
Objective	Unified framework for thermodynamic and transport properties	High-fidelity thermochemical and transport modeling for CFD and hypersonic flows	Fast LTE thermodynamic and transport property evaluation
Language	C++	C++	Python
User interaction model	C++ object-oriented scripting with structured workflow	C++ object-oriented interface with XML/database configuration	Python scripting
Extensibility	Type and class based modular design	Modular database-driven architecture	Script-level extensibility
Partition functions	Internal calculation	Database-driven (NASA, RRHO)	Hybrid (internal calculation and data loading)
Equilibrium composition	LTE and 2T Saha-based models	Constrained multiphase equilibrium solver (Database-driven)	Gibbs free energy minimization (LTE)
Collision integrals	Computed from multiple sources (potentials, cross sections, quantum phase shifts) and raw data loading	Database-driven (XML), limited to available studies	Empirical / approximate models
Thermodynamic properties	Computed from partition functions and composition	Database-driven and mixture averaged from NASA polynomials.	Computed from composition and partition functions
Transport properties	Devoto (LTE,2T), Zhang *et al.* (LTE, 2T)	Multiple transport models	Devoto (LTE)

can simply assign the new calculator to the objects through interface methods, like will be shown in the code (3).

Ultimately, a single `compute()` call on an `OmegaCalculator` and a `TcsCalculator` through `CollisionIntegral<T1,T2>` or on `ParfCalculator` through `PartitionFunction<T>` objects, called by the box classes for the overall sets of objects, will execute the correct algorithms. This ensures that the user-facing interfaces remain aligned with the physical abstraction of the problem, automating computations, and making the program easier, powerful, and more intuitive to use and extend.

3.4. Design trade-offs and performance considerations

While strategies like compile-time polymorphism are employed to reduce run-time overhead, the primary objective of the PPFM architecture is not to maximize raw computational performances, but to enhance modularity, extensibility, and usability. The object-orientated and template-based design introduces a moderate computational overhead compared to highly optimised legacy C or Fortran codes, which are often tailored to specific applications and rely on procedural implementations.

However, this trade-off enables a unified and flexible framework, where different stages of plasma property calculations - including partition functions, composition, collision integrals, and transport properties - are handled within a single consistent environment. In legacy implementations, these components are typically distributed between separate codes and workflows, requiring manual data handling and complex input/output management, as it did for the code used as a foundation for the implementation of PPFM [11]. PPFM removes this fragmentation by integrating all steps into a single coherent pipeline, improving reproducibility, and reducing user effort.

PPFM is therefore not intended to outperform highly optimized legacy solvers and existing tools such as MUTATION++ and minplascalc, but to provide a flexible and extensible research framework oriented to advanced models implementations, gathering together in a simple environment all the workflow and physical modelling required for thermophysical property determinations of plasmas, while ensuring new implementations with reduced efforts. This positioning is summarised in Table 1 with reference again to workflow Fig. 1.

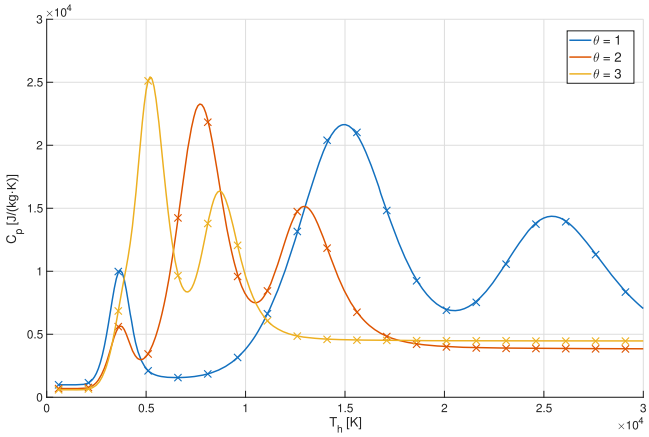


Fig. 3. Constant pressure specific heat for the ArH_2 mixture. 50%Ar, 50%H2. (Solid line -) PPFM. (x) Legacy C code used in [11].

4. Comparisons and validation

The following two sections are dedicated to comparisons and validation between PPFM computed values and results already published in the literature for an Argon-Hydrogen mixture [11] and pure Argon [30], using Devoto and Zhang et al. implemented theories, respectively. In addition, the corresponding `main.cpp` file used for the ArH_2 comparison is shown in the appendix code (3) at the end of this paper as an illustrative example showing the capabilities of the PPFM environment. Notice the simplicity within its syntax along with its operability and shortness.

4.1. Comparison with the ArH_2 mixture

An Argon-Hydrogen plasma mixture at atmospheric pressure is considered taking into account seven chemical species: Ar, Ar^+, Ar^{+2}, H, H^+, H_2, and electrons e^-. The transport properties are computed for values of the non-equilibrium parameters $\theta = 1, 2, 3$.

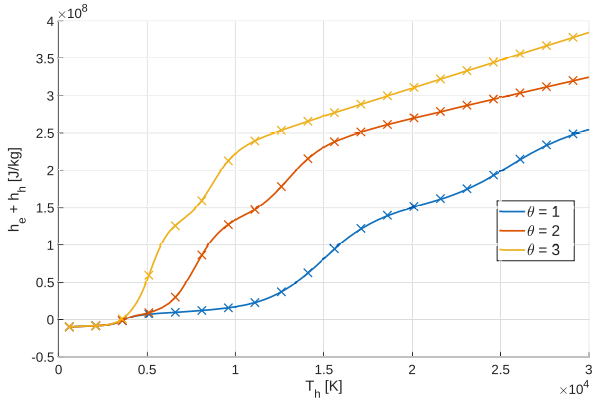


Fig. 4. Total specific enthalpy for the ArH_2 mixture. 50%Ar, 50%H2. (Solid line -) PPFM. (x) Legacy C code used in [11].

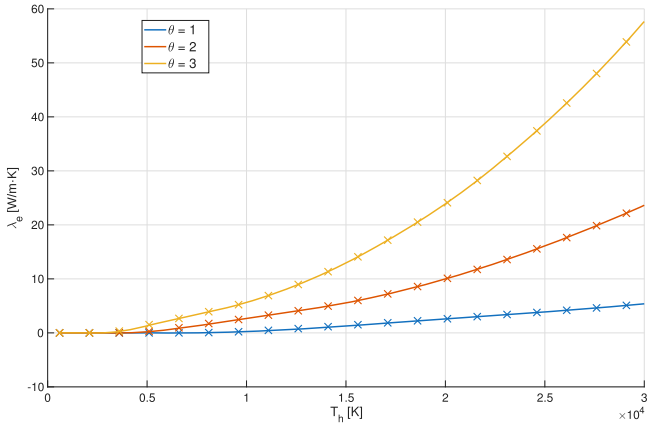


Fig. 5. Electron thermal conductivity for the ArH_2 mixture calculated with Devoto theory. 50%Ar, 50%H2. (Solid line -) PPFM , (x) Legacy C code used in [11].

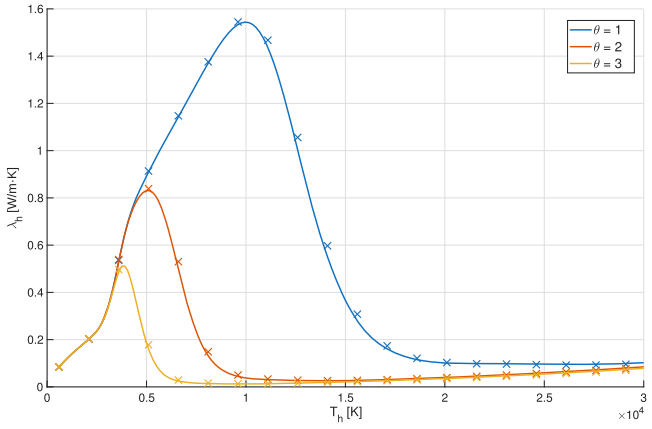


Fig. 6. Heavy particle thermal conductivity for the ArH_2 mixture calculated with Devoto theory. 50%Ar, 50%H2. (Solid line -) PPFM , (x) Legacy C code used in [11].

Tables 1, 3 and 4 of reference [11] have been faithfully reproduced through PPFM environment with minimal effort due to its powerful, modular implementation, and reproducible workflows. Previous computations of the legacy C code [9–11] required cumbersome modifications to multiple script-like files; as that version was implemented as a collection of procedural C functions organized in a serial execution structure, lacking the object-oriented framework that now characterizes PPFM.

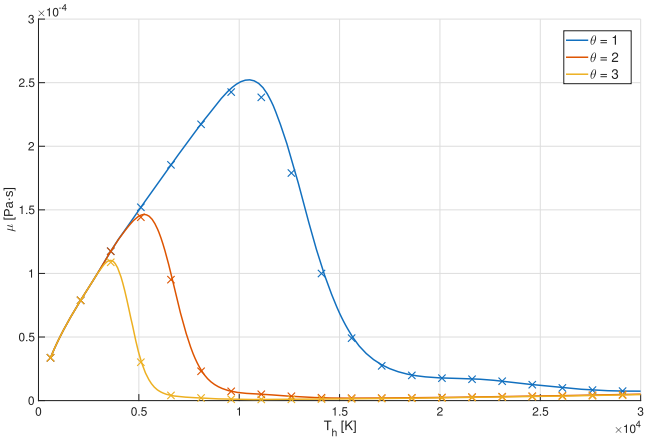


Fig. 7. Viscosity for the ArH_2 mixture calculated with Devoto theory. 50%Ar, 50%H2. (Solid line -) PPFM , (x) Legacy C code used in [11].

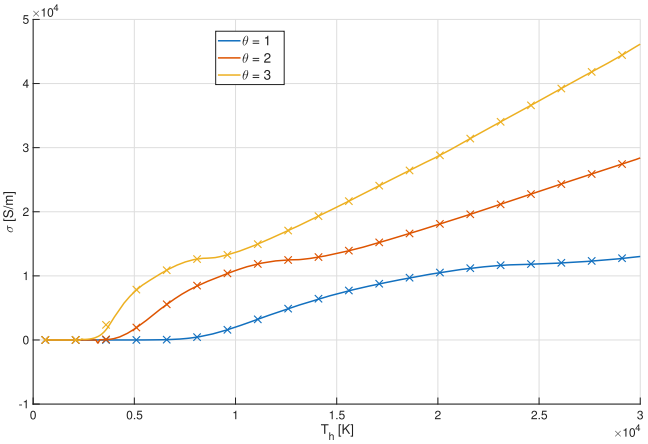


Fig. 8. Electrical conductivity for the ArH_2 mixture calculated with Devoto theory. 50%Ar, 50%H2. (Solid line -) PPFM , (x) Legacy C code used in [11].

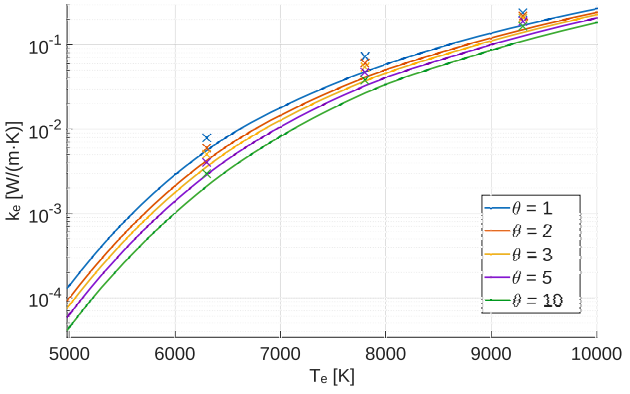


Fig. 9. Electrons thermal conductivity for the pure *Ar* calculated with Zhang et al. theory in logaritmic scale, closer look over temperature range 5000K to 10000K. (Solid line -) PPFM , (x) results reported in the appendix of [30].

Equilibrium two-temperature composition, thermodynamic properties, and transport properties have been computed, the transport properties are compared with the values calculated from the legacy C code used in [11] in Figs. 3–8.

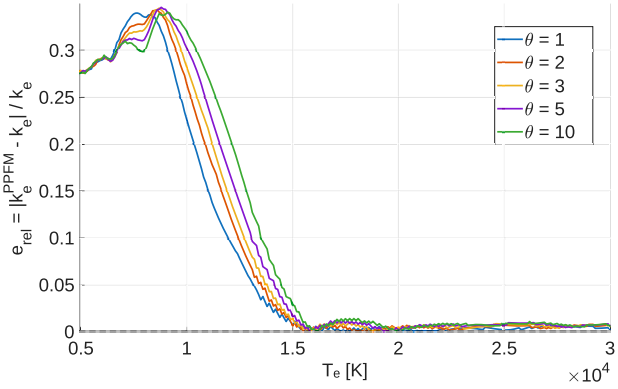


Fig. 10. Relative error between PPFM computed values and values reported in the appendix of [30], for the electrons thermal conductivities, in the temperature range 5000K to 30000K.

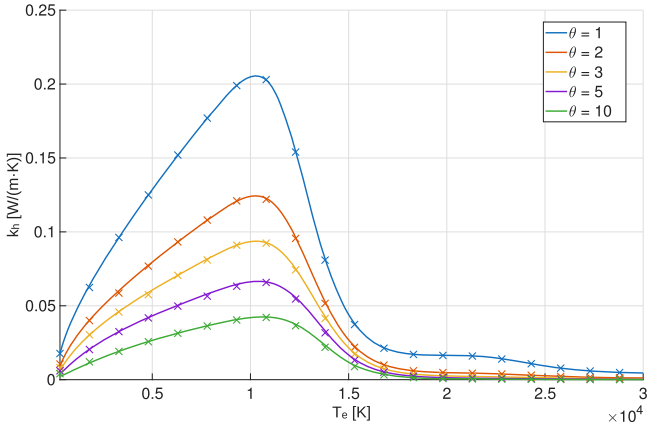


Fig. 11. Heavy particles thermal conductivity for the pure *Ar* calculated with Zhang et al. theory. (Solid line -) PPFM , (x) results reported in the appendix of [30].

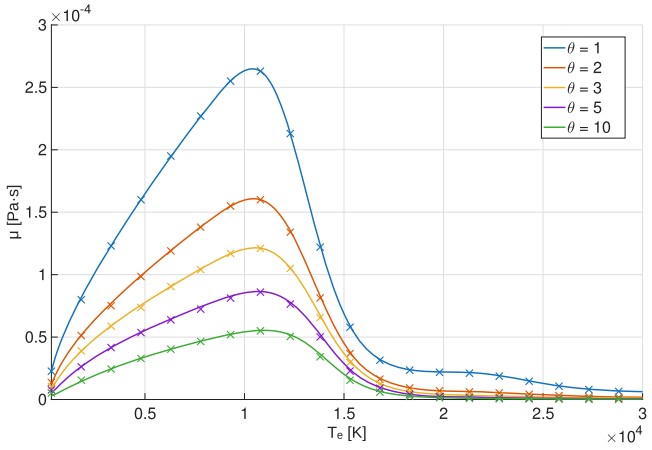


Fig. 12. Viscosity for the pure *Ar* calculated with Zhang et al. theory. (Solid line -) PPFM , (x) results reported in the appendix of [30].

4.2. Comparison with pure argon

The study presented in [30] was replicated using PPFM. A pure argon plasma at atmospheric pressure has been considered, taking into account four chemical species: Ar, Ar^+, Ar^{2+}, and e^-. Transport properties were computed for several non-equilibrium parameters i.e. $\theta = 1, 2, 3, 5, 10$, and compared with the data reported in the appendix of [30], as shown in Figs. 9–13. Some discrepancies can be observed in Fig. 9 for the electron thermal conductivity in the temperature range 6000–15000K, with PPFM computed values being lower than those of Zhang. As ionization increases, however, this disparity gradually vanishes as can be seen in Fig. 10 depicting the relative error of the two datasets. These differences may originate from different neutral species collision integrals, and differences in the composition calculation algorithm and implementation. Nevertheless, the PPFM results closely reproduce the overall trends reported in [30] for the electron thermal conductivity.

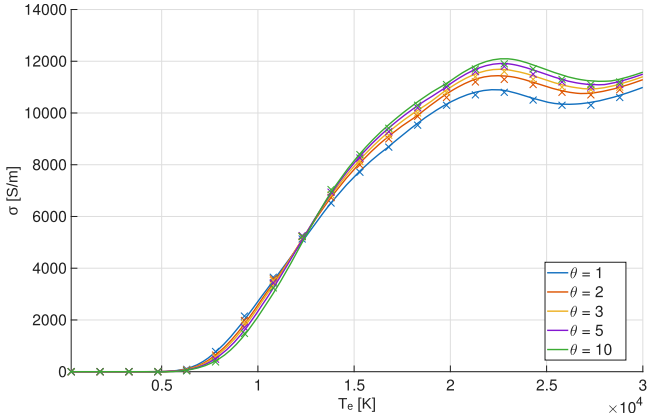


Fig. 13. Electrical conductivity for the pure *Ar* calculated with Zhang et al. theory. (Solid line -) PPFM , (x) results reported in the appendix of [30].

As in the previous case, the collision integrals for this calculation reported in [32] have been faithfully reproduced with relative ease thanks to the PPFM environment and capabilities. The corresponding `main.cpp` file used for this execution is available in the public demo folder of the program, available in his first version at Zenodo [33].

It is important to notice that comparisons with PPFM results allowed to find out an inconsistency affecting all transport properties for $\theta = 7$ in the dataset provided by [30]. Comparisons for such θ value are therefore omitted for clarity.[2] The Authors of [30] have been notified about this issue. The discovery of inconsistencies in literature data (just as previously done by one of the Authors in [11]) highlights even further the need for cross-code validations of properties data by different research groups, in a field that deals with complex algorithms and with the handling of huge amounts of heterogeneous atomic data, that may lead to error-prone codes. PPFM aims to facilitate by design the accessibility to such calculation also by non-specialized research groups, offering reproducible workflows to manage such calculations.

5. Conclusion

PPFM provides a solid foundation for future development and collaborative growth. Its architecture is designed to accommodate new physical models and algorithms, paving the way for PPFM to become a reference platform for investigating the thermodynamic and transport properties (TTP) of complex plasma mixtures across a wide range of conditions. At the same time, its structure makes it not only a practical tool for immediate research needs but also a sustainable framework, well suited for long-term maintenance, extension, and cross-disciplinary applications in fundamental plasma modeling studies.

The current implementation shows the potential of PPFM as a flexible and efficient environment for computing the thermodynamic and transport properties of partially to fully ionized plasmas under both equilibrium and non-equilibrium (2T) conditions, in a broad range of temperatures and pressures. Its modular object-oriented structure, reinforced by template metaprogramming and class hierarchies, allows researchers to model plasma mixtures with reduced effort, starting from

[2] PPFM values for $\theta = 7$ are available as supplementary material to this article.

microscopic descriptions up to macroscopic properties. Within this unified framework, complex effects such as charge exchange, excited states, and non-ideal interactions are consistently supported, enabling reproducible workflows and reliable outcomes for the plasma modeling community.

The results for the ArH_2 mixture confirm the reliability of the new Object-Oriented implementation, reproducing the results of the validated legacy C code. Likewise, the pure argon properties obtained with Zhang's et al. theory show good agreement with the data in [30]. This consistency strengthens confidence in the theoretical and numerical bases of PPFM, making it a reliable starting point for its application to more complex or previously unexplored plasma systems, provided appropriate database support is available.

Finally, the semantic separation embedded in the design of PPFM plays a central role in improving the clarity, maintainability, and extensibility of the software. By assigning clear responsibilities to each physical or computational entity, the framework mirrors the structure of the physical problem itself, minimizing conceptual overlap. This clarity enhances maintainability and extension, while supporting the adoption of PPFM as a reliable and user-friendly platform for plasma studies. The main program example reported in the appendix further highlights how tasks such as computing TTP with the Chapman-Enskog modified approach can be carried out in a straightforward and coherent manner, reinforcing the transparency and accessibility of the entire framework.

Software availability

The PPFM software, version 0.1.0, is publicly available through the Zenodo repository [33]. The source code is released under the Creative Commons BY 4.0 license and is actively developed in the public GitHub repository [34].

CRediT authorship contribution statement

A. Vagnoni: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization; **M. Gherardi:** Revision; **E. Ghedini:** Revision, Conceptualization, Software.

Data availability

Data will be made available on request.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Alberto Vagnoni reports financial support was provided by European Union. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This work was supported by the Italian National Recovery and Resilience Plan (PNRR), funded by the European Union under the program NextGenerationEU.

Appendix A

```
/*

    MAIN FILE USED IN THE VALIDATION PROCESS
    DESCRIBED IN THE ILLUSTRATIVE EXAMPLE OF

    "PPFM (Plasma Properties For Many): An Object Oriented C++
    Library for Computing Thermodynamic and Transport Properties
    of Plasmas Under Different Operating Conditions"

    by

    A.Vagnoni, E.Ghedini, M.Gherardi

    The scheme for transport cross sections and
    collision integrals computation described in the tables of

    "Two-temperature thermodynamic and transport properties
    of argon{hydrogen and nitrogen{hydrogen plasmas"

    by

    V Colombo, E Ghedini and P Sanibondi

    DOI : doi:10.1088/0022-3727/42/5/055213

    Has been faithfully reproduced.

*/
#include "GasMixture.h"
#include "CiBox.h"
#include "PartitionFunction.h"
#include "Devoto.h"
#include "Thermodynamics.h"
#include "ZhangMurphyTP.h"
#include "Potential.h"
#include "CollisionIntegral.h"
#include "DataPrinter.h"

int main() {

    // Output folder
    std::string folder = "demo/ArH2_NLTE";

    // GasMixture definition
    GasMixture* mix = new GasMixture (

        300. , 101325. ,
        new Argon , new ArgonI , new ArgonII ,
        new Hydrogen , new HydrogenI , new MolecularHydrogen ,
        new Electron

    );
```

Listing .1. Source code of the main file used in the production of the datasets for the Argon Hydrogen mixture validation.

```
// Non equilibrium parameters to loop on
std::vector<double> NEparam = { 1. , 2. , 3. };

// Set temperature range to compute on
std::vector<double> T = arange ( 300., 30100., 100. );

// Setting editable partition function box
auto editableQbox = mix->getCompositionObj()->getPfBox();

// Get info on partition functions for editing
editableQbox->info();

(*editableQbox)[0]->setAbInitio();
(*editableQbox)[1]->setAbInitio();
(*editableQbox)[2]->setAbInitio();
(*editableQbox)[3]->setAbInitio();
(*editableQbox)[5]->setTable();

// Set desired molar fraction
mix->setMoleFractions ( { 0.5, 0.5 } , new Argon, new MolecularHydrogen );

// Editable Collision Integrals Box
CiBox cibox (mix);

// Get info on s for editing
cibox.info();

// Species polarizabities
double alphaAr = 1.641100; // Ang^3
double alphaH = 0.666793; // Ang^3
double alphaH2 = 0.804225; // Ang^3


// Ar - Ar
cibox[0]->TCScalculator = new AdaptChiIntegrator (

    cibox[0]->GetIntInterface(),
    new HFDTCS2_ArAr()

);

// Ar - Ar+
cibox[1]->Load(false);
auto arari = cibox[1]->GetIntInterface();

// Elastic + Inelastic collision
cibox[1]->TCScalculator = new CsHolder(

    // Elastic integration of the potential
    new AdaptChiIntegrator ( arari,

        new HulburtHirschfelderUnreduced (

            // eps , re , we , weXe, Be , Alphae
            0.01136, 3.759e-10, 3.068e-3, 256., 5.973, 0.39
```

```
        )

    ) ,

    // Two different Charge Transfer Cross Sections
    new MultiCs (
        arari, {

            new ChargeTransferCs ( arari, 26.39, 1.12 ),
            new ChargeTransferCs ( arari, 18.96, 0.83 )

        } , {

            2,
            4
        }
    )

);

// Ar - Ar+2
cibox[2]->Pot( new Polarization(new Argon,new ArgonII,alphaAr));

// Ar - H
cibox[3]->Pot( new Morse2Param ( 138.3 , 2.783 ) );

// Ar - H+
cibox[4]->Pot( new Morse3Param ( 4.1 , 1.792 , 1.33 ) );

// Ar - H2
cibox[5]->Pot( new Morse3Param ( 7.019e-3 , 1.895 , 3.615 ) );

// Ar - e-
cibox[6]->Load(false);
InteractionInterface* Are = cibox[6]->GetIntInterface();

// Different Cross Sections for different energy ranges
cibox[6]->TCScalculator = new ThresholdCs ( Are,
    {
        // Quantum phase shifts from literature
        new PhaseShiftsLoader ( "Bell1984" , Are ),
        new PhaseShiftsLoader ( "Gibson1996" , Are ),

        // Differential cross sections
        new DcsLoader(Are)
    },
    {
        1., // Bell until 1.eV
        10. // Gibson until 10.eV,
              // Dcs after.
    }
);

// Ar+ - H
// Multi-state interaction with different potentials
```

```
cibox[9]->MultiPot (
    {
        // Re, D, B0, gamma, lambda
        new Morse5Param ( 1.3229, 2.1806, 1.509, -0.4788, 0.13659 ),
        new Morse5Param ( 1.3229, 2.2873, 2.118, 0.21770, 0.21200 ),
        new Morse5Param ( 1.3229, 1.8190, 1.613, -0.5157, 0.14108 ),
        new Morse5Param ( 1.3229, 4.0051, 1.080, -0.4395, 0.15825 )
    },
    {
        // statistical degeneracies gk of the states
        2.,
        3.,
        6.,
        1.
    }
);

// Ar+-H2
cibox[11]->Pot(
    new Polarization ( new ArgonII , new MolecularHydrogen, alphaH2 )
);
// elastic + inelastic
cibox[11]->LoadInelastic();

// Ar+2 - H
cibox[14]->Pot (
    new Polarization ( new ArgonII, new Hydrogen, alphaH )
);



// Ar+2 - H2
cibox[16]->Pot (
    new Polarization ( new MolecularHydrogen, new ArgonII, alphaH2 )
);

// H - H+
cibox[19]->Load(false);

// getting the interface
InteractionInterface* i = cibox[19]->GetIntInterface();

cibox[19]->TCScalculator = new CsHolder (

    // Elastic collision
    new MultiCs( i,
    // two states for the elastic collision
    {

        new AdaptChiIntegrator ( i, new Morse2Param ( 51.75 , 1.677 ) ) ,
        new ThresholdCs ( i,
        {
            // Potential BEFORE 10. eV
            new AdaptChiIntegrator ( i, new PowerPot ( -282. , 5.8 )) ,

            // Potential AFTER 10. eV
```

```
            new AdaptChiIntegrator ( i, new PowerPot ( -18.76 , 3.47 ))
        },
        {
            // Threshold energy
            10.
        }
        )
    } , {
        // statistical degeneracies gk of the states
        2.,
        2.
    }
    ),
    // Inelastic collision - Charge Transfer
    new ChargeTransferCs ( i, 28.69, 1.3 )
);


// H-e- Integration of Momentum Transfer Cross Section
cibox[21]->Load(false);
// Load the momentum transfer cross section for H-e- interaction
cibox[21]->LoadElastic();

// H+-H2
cibox[23]->Pot(

    new Polarization ( new HydrogenI , new MolecularHydrogen , alphaH2 )

);
cibox[23]->LoadInelastic();

// H2-e- Integration of differential scattering cross sections
cibox[26]->Load(false);
cibox[26]->LoadDCS();

// Output modules initialization
Thermodynamics thSolver;
ThermodynamicsCsv th(&thSolver, folder+ "/Thermodynamics" );

// NOTE: pass explicit solvers to CSV printers
DevotoTpCsv Dev ( new DevotoTP(&cibox), folder+ "/Devoto" );
ZhangTpCsv zm ( new ZhangMurphyTP(&cibox), folder+ "/Zhang" );

// You can confirm the editing of the Collision Integrals and Partition Functions if you want.
editableQbox->info();
cibox.info();

// cibox.PrintDeflectionAngles ( T, mix, folder+ "/Deflection Angles" );
cibox.PrintTransportCrossSections ( T, mix, folder+ "/Transport Cross Sections" );
cibox.PrintCollisionIntegrals ( T, mix, folder+ "/Collision Integrals" );

// Loop on NEparam
for (size_t i = 0; i < NEparam.size(); i++) {

    // NEparam set to Mixture object
    mix->theta->set(NEparam[i]);
```

```
            // int to write filenames
            int str = NEparam[i];

            // Computing and printing data
            th.Print ( "Theta" + std::to_string(str), T, mix );
            Dev.Print ( "Dev1966_Theta" + std::to_string(str), T, mix );
            zm.Print ( "ZM2013_Theta" + std::to_string(str), T, mix );

        }

    return 0;

    }
```

References

[1] M. Boulos, P. Fauchais, E. Pfender, T.E. Magin, J. Heberlein, Handbook of Thermal Plasmas, Springer, 2023.
[2] E. Ghedini, P. Sanibondi, V. Colombo, J. Phys. D Appl. Phys. 43 (2010) 435203.
[3] S. Chapman, T.G. Cowling, The Mathematical Theory of Non-Uniform Gases, Cambridge University Press, 3rd ed., 1970.
[4] G. Colonna, A. D'Angola (Eds.), Plasma Modeling Methods and Applications, IOP Publishing, 2nd ed., 2022.
[5] A. Gleizes, Y. Cressault, P. Teulet, Plasma Sources Sci. Technol. 19 (2010) 055013.
[6] J.B. Scoggins, V. Leroy, G. Bellas-Chatzigeorgis, B. Dias, T.E. Magin, SoftwareX 12 (2020) 100575.
[7] Q.G. Reynolds, B. Bowman, M.W. Erwee, I.J. Geldenhuys, C. Sandrock, G.A. Venter, B.S. Xakalashe, J. Zietsman, J. S. Afr. Inst. Min. Metall. 125 (2025) 129.
[8] Q. Reynolds, minplascalc, 2023, https://github.com/quinnreynolds/minplascalc.
[9] V. Colombo, E. Ghedini, P. Sanibondi, Prog. Nucl. Energy 50 (2008) 921–933.
[10] V. Colombo, E. Ghedini, P. Sanibondi, The effects of flow rate and gas mixture on the performance of a microwave plasma torch, Plasma Sources Sci. Technol. 20 (035003) (2011).
[11] V. Colombo, E. Ghedini, P. Sanibondi, J. Phys. D Appl. Phys. 42 (2009) 055213.
[12] J. Aubreton, V. Rat, M. Elchinger, P. Fauchais, A.B. Murphy, Plasma Chem. Plasma Process. 3 (1983) 369.
[13] A.B. Murphy, J.P. D. A.P. Murphy, J. Phys. D Appl. Phys. 33 (2000) 314.
[14] A.B. Murphy, Plasma Chem. Plasma Process. 21 (2001) 557.
[15] A. Laricchiutta, D. Bruno, M. Capitelli, C. Catalfamo, R. Celiberto, G. Colonna, P. Diomede, D. Giordano, C. Gorse, S. Longo, D. Pagano, F. Pirani, Eur. Phys. J. D 54 (2009) 607–612.
[16] A. Laricchiutta, G. Colonna, D. Bruno, R. Celiberto, C. Gorse, F. Pirani, M. Capitelli, Chem. Phys. Lett. 445 (2007) 133–139.
[17] F. Pirani, M. Albertí, A. Castro, M.M. Teixidor, D. Cappelletti, The role of the van der Waals interactions in the adsorption of polar molecules on metallic surfaces, Chem. Phys. Lett. 394 (1–3) (2004) 37. https://doi.org/10.1016/j.cplett.2004.06.067
[18] F. Pirani, G. Maciel, D. Cappelletti, V. Aquilanti, Int. Rev. Phys. Chem. 25 (1–2) (2006) 165.
[19] A.B. Murphy, J. Phys. D Appl. Phys. 44 (2011) 285203.
[20] M. C. M. van de Sanden, P. P. J. M. Schram, A. G. Peeters, J. A. M. van der Mullen, G. M. W. Kroesen, Phys. Rev. A 40 (1989) 5273.
[21] D. Godin, J.Y. Trépanier, Plasma Chem. Plasma Process. 24 (2004) 447.
[22] G. Colonna, A. Laricchiuta, Comput. Phys. Commun. 178 (2008) 809.
[23] B. Barker, D. Fock, F.J. Smith, Phys. Fluids 6 (1963) 1345.
[24] R.S. Devoto, Phys. Fluids 10 (1967) 354.
[25] J.H. Ferziger, H.G. Kaper, Mathematical Theory of Transport Processes in Gases, Stanford University Press, 1972.
[26] R.S. Devoto, Phys. Fluids 9 (1966) 1230.
[27] R.S. Devoto, Phys. Fluids 10 (1967) 2105.
[28] X.-N. Zhang, H.-P. Li, A.B. Murphy, W.-D. Xia, Phys. Plasmas 20 (2013) 033508.
[29] V. Rat, P. André, J. Aubreton, M.F. Elchinger, P. Fauchais, A. Lefort, Phys. Rev. E 64 (2001) 026409.
[30] X.N. Zhang, H.P. Li, A.B. Murphy, W.D. Xia, Plasma Sources Sci. Technol. 24 (2015) 035011.
[31] B. Stroustrup, The C++ Programming Language, Addison-Wesley, 4th ed., 2013.
[32] A.B. Murphy, C.J. Arundell, Plasma Chem. Plasma Process. 14 (1994) 451.
[33] A. Vagnoni, E. Ghedini, Zenodo, 2025, https://doi.org/10.5281/zenodo.17191784.
[34] A. Vagnoni, E. Ghedini, PlasmaModelling/PPFM GitHub repository, 2025, https://github.com/PlasmaModelling/PPFM.