Universidad Carlos III de Madrid

Bachelor's Degree in Aerospace Engineering

Bachelor Thesis

IMPLEMENTATION AND CHARACTERIZATION OF AN OPEN-SOURCE PLASMA PLUME CODE

Pablo Moreno de Santos

Supervised by:
Mario Merino

Leganés, May 2016

Dedicated to my family,
and friends

Abstract

An open-source code for the far region plasma plume expansion into vacuum is presented. The derived non-magnetic and colisionless plasma model is solved under two semi-analytical solutions valid hypersonic plumes: The Self-Similar Method and the Asymptotic Expansion Method.The validity, and deficiencies of each approximation method are presented, and the accuracy of the results is further investigated and compared against the more accurate AEM reinitialization marching scheme: The relative errors in plume density and velocity are shown to increase axially and radially, in the order of $10^{-2} - 10^{-3}$*. A parametric study on the plume far field divergence angle, and ambipolar electric field is carried out, due to its importance within the space propulsion framework.The entire model is implemented in the Python programming language, and made available to the public in specific web repositories. A discussion on the aspects of developing such open-source codes as well as the steps followed during the process, are detailed. Lastly, a qualitative discussion on the limitations of the numerical model and the secondary plasma plume theory behind it, is offered.*

Acknowledgements

I would like to thank in Prof. Mario Merino for the opportunity to complete this dissertation project, for his patience and for the assistance he has offered me during the project.

My most sincere gratitude goes also to all my friends, for all their support, help and cheering during sleepless nights.

Finally, I would like to express my warmest and deepest gratitude to my family, for their countless smiles and advices in times of help, for their encouragement and assistance during all my life and the duration of this project, and for guiding me through the worst moments.

To all of you, thank you.

Contents

List of Figures

Chapter 1

Introduction

The understanding of the universe, the need to reach distant stars and galaxies far beyond our Solar System and ultimately, the search of extraterrestrial life have made rocket propulsion systems and more particularly electric propulsive techniques, a field of active search and development in deep space exploration.

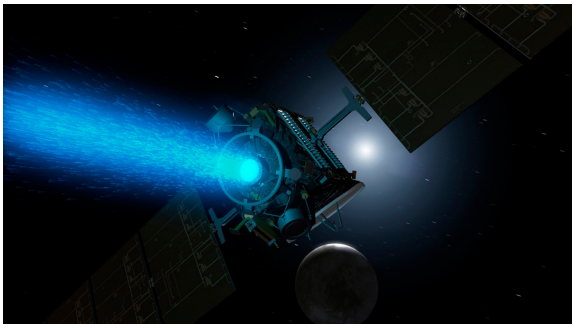


Figure 1.1: Visual characterization of the plasma plume expelled from a satellite

Rocket propulsion systems are often classified according to the type of energy source backing its operation. In this manner, electric propulsion systems rely on electric power sources (nuclear, solar radiation receivers,batteries) whereas chemical propulsion systems obtain the energy from high temperature thermodynamical combustion processes of propellant chemicals. As a consequence, while chemical propulsion is found to be "energy limited" due to the fixed amount of energy stored in the chemical reactants per unit-mass, electric propulsion systems are termed as "power limited": the rate at which the energy is extracted from the external power source and supplied to the propellant is limited by the mass available for the power system. Another distinction between chemical propulsion and electric propulsion is found in the method of producing thrust. Thrust F is the force produced by a rocket propulsion system acting upon, the reaction experienced by its structure due to the ejection of matter at high velocity. Roughly speaking, Thrust due to change in momentum is expressed as:

$$F = \frac{dm}{dt} v_2 = \dot{m} g_0 I_{sp},$$

where v_2 is the effective exhaust velocity and I_{sp} is the specific impulse of the exhaust propellant. The effective velocity v_2 accounts for any pressure contribution effects to thrust F, in the previous equation. The specific impulse is defined as the total impulse of the rocket propulsion system I_t, per unit weight of propellant

$$I_{sp} = \frac{I_t}{g_0 m} = \frac{\int_0^{t_b} F dt}{g_0 \int m dt}.$$

Both F, and I_{sp} are important figures in the performance analysis of rocket propulsion systems which serve to further differentiate chemical propulsion systems, from electrical ones. Being "en-

ergy limited", chemical propulsion systems achieve lower values of I_{sp} or v_e but nonetheless, since the rate at which energy can be supplied to the propellant is independent of its mass, very high thrust levels can be achieved. On the other hand, electric propulsion systems being "power limited", attain low values of thrust but due to the large amount of energy which can be delivered to a given mass of propellant, v_2 and I_{sp} can be much larger than that available from a chemical propulsion system. Figure 1 shows typical performance parameters for different types of rocket propulsion systems.

Deepening into electric propulsion systems, different types can be distinguished between: *elec-*

Engine Type	Specific Impulse[a] (sec)	Maximum Temperature (°C)	Thrust-to-Weight Ratio[b]
Chemical—solid or liquid bipropellant	200–468	2500–4100	10^{-2} – 100
Liquid monopropellant	194–223	600–800	10^{-1} – 10^{-2}
Nuclear fission	500–860	2700	10^{-2} – 30
Resistojet	150–300	2900	10^{-2} – 10^{-4}
Arc heating—electrothermal	280–1200	20,000	10^{-4} – 10^{-2}
Electromagnetic including pulsed plasma (PP)	700–2500	—	10^{-6} – 10^{-4}
Hall effect	1000–1700	—	10^{-4}
Ion—electrostatic	1200–5000	—	10^{-6} – 10^{-4}
Solar heating	400–700	1300	10^{-3} – 10^{-2}

Figure 1.2: Performance Parameters for different Rocket Propulsion Systems [1]

trothermal, where the gaseous propellant is heated electrically and expanded thermodynamically to supersonic speeds through a nozzle, similarly to chemical rockets; *electrostatic*, where gas propellant acceleration is achieved by the interaction of electrostatic fields on non-neutral propellant particles such as atomic ions; and *electromagnetic*, where gas propellant expansion is generated by the interaction of electric and magnetic fields within a plasma. Figures 1.3 and 1.4 show typical performance parameters, and the potential mission utility regions of various types of electrical propulsion systems.

Type	Thrust Range (mN)	Specific Impulse (sec)	Thruster Efficiency[a] (%)
Resistojet (thermal)	200–300	200–350	65–90
Arcjet (thermal)	200–1000	400–1000	30–50
Ion thruster	0.01–500	1500–8000	60–80
Solid pulsed plasma (PPT)	0.05–10	600–2000	10
Magnetoplasma dynamic (MPD)	0.001–2000	2000–5000	30–50
Hall thruster	0.01–2000	1500–2000	30–50
Monopropellant rocket[b]	30–100,000	200–250	87–97

Figure 1.3: Performance Parameters for different Electrical Propulsion Systems [1]

Although electrical propulsion systems have been successfully implemented in numerous satellites and space missions since their advancement in the late 1960s, the technology behind these electrical thrusters which is proven to be reliable and offers higher specific impulse with lower costs, is still in need of lesser-known performance analysis and corrections. The characterization of the high energy exhaust from an electrical propulsion thruster known as the plume, and its impact over surrounding elements in the spacecraft is one of such plasma propulsion technology fields in

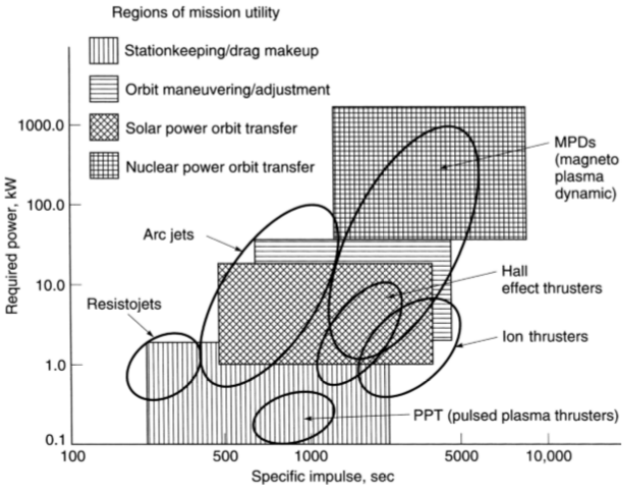


Figure 1.4: approximate regions of application of different electrical propulsion systems in terms of power and specific impulse. [1]

need of improvement, and the central topic of this dissertation.

1.1 State-of-the art in Plasma Plume Characterization

The expansion of the plasma plume outside the thruster is a field of continuous study, primarily due to the negative effects of plume divergence on the spacecraft. Concisely, higher plume divergence leads to greater propulsive losses, as part of the impulse is directed radially rather than axially. Besides, the impact of the high energy particles immersed inside the plasma, with surrounding structural components in the spacecraft such as solar arrays, is more likely to occur when the plume divergence increases. The erosion and contamination of impacted surfaces that follows, known as sputtering, severely reduces satellite life. Finally, there exists high risk of electric arc discharging between equipment in the spacecraft at different electric potential, in the presence of the conducting plasma in the plume. Thus, the need of exploring the physical effects of plasma plume expansions in detail.

The subject matter has been progressively studied after exploratory investigations in stationary plasma thruster plumes (STP) [7], [8], a wide acceleration zone thruster precedent of the Hall-Effect Thruster (HET) and developed in the Soviet Union during the early 60's. Although preliminary, this studies did an exemplary job in identifying the detrimental effects and key plasma parameter of plume expansions, using probes at specific distances from the thruster exit. In fact, these analysis even differentiated between the *near-plume* and *far-plume* regions as a qualitative division in the plume expansion model, to study the behavior of electron density and ion current densities, plasma temperature and ion energy variables at several divergence angles from the plume axis and downstream distances from the thruster exit. Figure 1.5 shows the main findings of such studies. Precisely, based on this distance from the thruster exit, the previously cited plasma properties and other physical factors which control the expansion (ion kinetic energy, plasma pressure or even external electric and magnetic currents) have different weight. Up to a few thruster radii, the complex *near-region plume* is found. In this region the plasma is heterogeneous, and the effects of local electric fields (such as the potential well at the last grid in Gridded Ion Thrusters (GIT) or the magnetic field induced in Hall Effect Thrusters), and charged particle collisions dominate due to the presence of a dense cloud of neutral particles.On the contrary, in the *far-region plume* the plasma is characterized by being quasi-neutral, collisionless and carrying no current. In this region plume divergence increases up to significant values (10^{o}-20^{o} deg in GIT or 40^{o}-50^{o} deg in HET). Multiple plume characterizations have been done subsequently to these preparatory studies, most of them particularizing in the analysis of Hall-effect thruster plumes.HET are particularly recognized

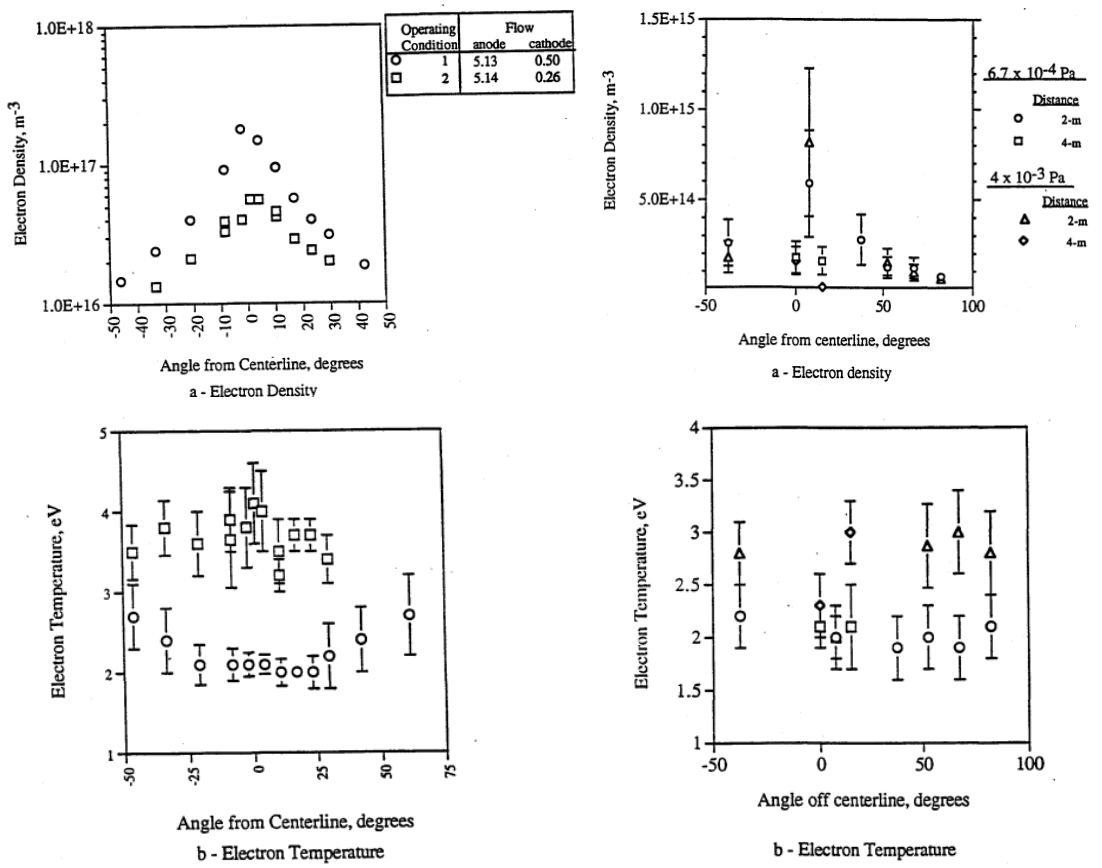


Figure 1.5: Preliminary studies on plasma plume expansions for SPTs [7].Particularly, important plasma variables, like electron density and temperature, are characterized in the near expansion region (plots (a) and (b)), and the far-region plume (plots (c) and (d) over the plume divergence angle.

as one of the most promising propulsion systems due to their compactness and simplicity over GIT´s offering the same thrust. However, the exhaust plume is expelled at large divergence angles form a HET.Therefore, knowledge of the plasma plume evolution is fundamental in order to asses the interference and integration risks between the propulsive system and surrounding environment as well as systems in the spacecraft. Two approaches in plume investigation have been clearly distinguished between the experimental in-flight or laboratory characterization, and the theoretical description. Generally speaking, *near-plume region* diagnosis is tested in-flight, in laboratory experiments or otherwise using advanced numerical tools able of tracking the numerous physical complexities, such as advance Particle-In-Cell codes. The *far-region plume* is otherwise investigated with simpler two fluid models with semi-analytical approximations such as the Self-Similarity plasma plume models (SSM). The two trends are clearly presented in [13] and [14].The former research presents the characterization of the near-field plume expansion variables under a Hybrid-PIC simulation,the outcome being described in figure 1.6 plot (a). Conversely,research conducted in [14] examines the far-field plume by means of a comparison between experimental plasma properties with the semi-analytical SSM fluid model.The model was proven to accurately match experimental data with little error as depicted in figure 1.6 plot (b). Ultimately,some authors went as far as to compare the accuracy between both approaches, supporting the validity and accuracy of the results[15]

Due to the current limited availability of in-flight plume data in the space environment, experimental ground tests have been performed although their cost is still very high. Moreover, ground tests completed in vacuum chambers, fail to return accurate results in part due to the limited sizing of the chamber, and the build-up of residual gas pressure. The divergence between plume

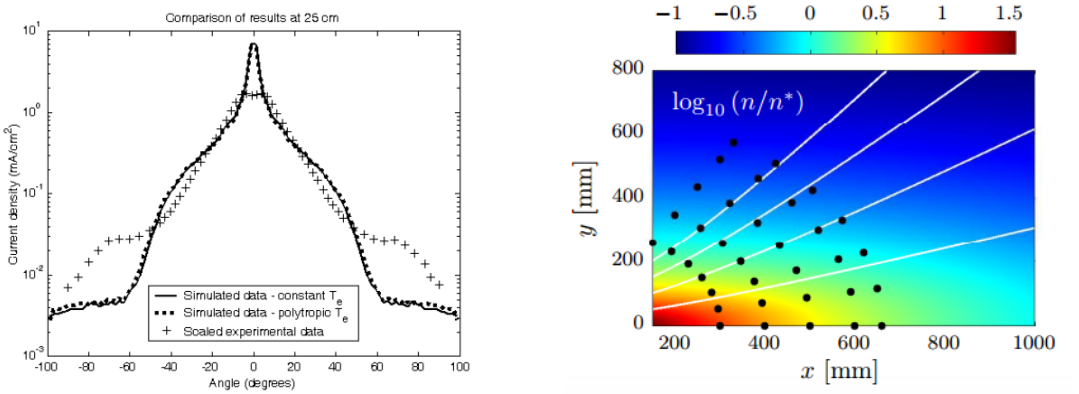


Figure 1.6: Comparison between experimental results in Space and in vacuum chamber ground tests, plot (a) [14], [13]. Plot (b) lays out experimental data values (black dots) over the semi-analytical SSM density map, showing akin results. [14].

expansion in Space and in vacuum chamber laboratory tests is fully detailed in [22]. In particular, this study concluded that the divergence of plumes is greater in space conditions and so is the ion current density and electron temperature in the periphery as shown in figure 1.7.

On the other hand,*Far-region plume* numerical models are in good agreement with experimental tests yet providing a quick and clear understanding of the physics involved in the problem. Therefore, ground and in-flight experiments are more commonly used to provide finer initial data tuning for such models, while theoretical PIC and semi-analytical two-fluid models propagate this experimental data downstream to imitate space conditions.

All the research presented up to this point has been focused towards the identification and characterization of plumes for their negative impact on surrounding elements. However, in recent years the vast and booming amount of space debris, old and defunct man-made satellite waste in LEO (Low earth orbits) and GEO (Geostationary Earth Orbits) around the earth, has pushed for new techniques in active space waste mitigation and removal. Such is the case in [2], where the novel concept of the Ion Beam Shepherd (IBS) is presented as an alternative to existing methods

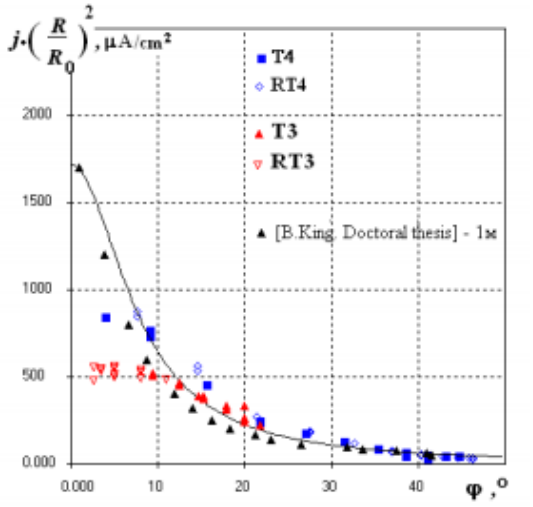


Figure 1.7: Evolution of current density over plume divergence for experimental in-flight data of T4-R4 and R3-T3 pair of thrusters in STP's (symbols), and ground laboratory fitting model of data (solid curve) [22].

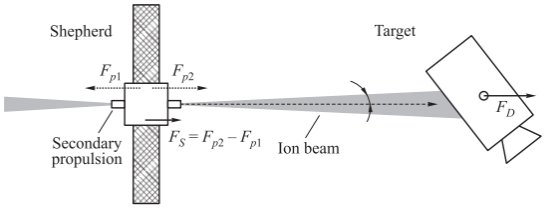


Figure 1.8: Schematics of the forces involved in the process of deorbiting space debris within the IBS concept [2].

like docking maneuvers. The new concept, takes special advantage of plasma plume physics, and aims to reposition space debris safely to lower and unoccupied orbits by exploiting the momentum transmitted by a collimated beam of quasi-neutral plasma impinging against its surface. In any case, the plume was studied with a SSM model to gain quick understanding on the forces and torques exerted by the shepherd satellite on the space debris. The results presented were promising, showing momentum transfer efficiencies close to unity between the two bodies proving therefore, the IBS concept to be effective deorbiting debris.

1.2 Objectives of the present work

This dissertation is in nature computational, and studies the physics behind the expansion of such plasma plumes into vacuum, particularly in the far-region domain, and presents the two revised semi-analytical models in Python programming language. The first numerical method introduced in this dissertation is the Asymptotic Expansion Method (AEM). In addition, based on the self-similarity concepts introduced in Refs [2],and [3], an alternative Self-Similar Method approach is presented as solution the expansion problem. It must be noted that a preliminary software tool based on these two approaches was presented in [4]. This previous code suite was developed in Matlab, a powerful high-level programming language similar to Python, but lacked of the appropriate debugging and testing framework before its open distribution to the scientific public. This dissertation presents therefore, the results of an open-source plasma plume code developed in Python, with a clear object-oriented structure, that is more readable and freely available to the academic community. It is of paramount importance that academic code is questioned not only in terms of accuracy, but also in terms of user interface and software integrity. Thus, special attention has been placed in the interface testing and debugging of the Python software, as means to completely validate the syntax structure of the plume code. This critical step serves as a further validation check in the process of developing scientific code, reinforcing its completeness before public use. Nevertheless, the new Python results are also tested and validated against the previous Matlab version outcome, in the attempt to clarify dissimilarities between the codes and areas of future development. The plume expansion Python code, its accompanying documentation in *PDF* and *HTML* format, as well as the testing suite and additional information are enclosed under the *HyperPlume* project. This dissertation presents the structure of such project, and its addition as an Open-Source web repository in GitHub.

This dissertation also summarizes quickly the need and tools used in the process of developing Open-Source software. By providing accessible Python code to all academic and non-scholar community freely, the exchange of information, revision of code, and fast as well as deep understanding physical phenomena in hypersonic plasma plume expansions is granted. Python is an interpreted open source programming language, used for its simplicity and its powerful capabilities. Its high-level built in data structures makes it an attractive choice for standalone application development as well as for as a scripting language to connect existing components together. Moreover, the code presented in this dissertation follows an object oriented programming approach, where the user is able to create new Plume objects that support inheritance and code reuse. Furthermore, the benefits of introducing Object-Oriented and test-driven programming approaches in scientific code, for validation and checking purposes is commented.

Finally, and back to the topic of plume expansion analysis and characterization, important parameters such as plasma density, plasma temperature are investigated in carefully latter sections, to fully characterize the physics of the far-region plume and check the validity of the model. Additionally, other important concepts in plume expansions, such as the generation of ambipolar electric fields and potentials in plasma or the far field divergence evolution of the plume, are detailed to provide physical insight in the model. Finally, relevant secondary phenomena in plasmas that were neglected in the assumptions of the presented expansion model, are also discussed qualitatively to explain at least the limitations of such model.

1.3 Dissertation Structure

The structure of the dissertation is as follows. In Chapter 2, a complete description on the physics of the cold plasma plume expansion model is presented.In Chapters 3 and 4, the two semi-analytical methods, the Self-Similar Method and the Asymptotic Expansion Method are detailed and discussed. In addition, an exhaustive description on the Open-Source Code with its format, implementation, as well as documentation and differences with previous plume software is given in Chapter 5. A complete characterization of different plasma variables such as plasma density, velocity or temperature for different AEM and SSM expansion plumes is addressed independently for each method, in chapter 6. In this chapter, a descriptive comparison between the two models is issued as well. Finally, a detailed discussion on remaining plume expansion parameters such as the ambipolar electric potential or the far-field plume divergence angle, and the secondary physical processes occurring in plasmas during plume expansion is presented in chapter 7.

Chapter 2

Physics and modeling of the far-region plasma plume

As opposed to the complex near plume region just outside the thruster exit, where the plasma is highly non-homogeneous and dominated by 3D effects such as the neutralizer electron flux, the potential well outside a Gridded Ion Thruster or the magnetic field induced in the Hall Effect thruster, the far-region plume is quite simpler.
The far region plasma,which extends from a distance of approximate 0.6-1m outside of the thruster, is typically collisionless and non-magnetic. Furthermore, the main driving parameters in the plasma are the ion flux, the electron pressure and the am-bipolar electric field. Thus, the steady-state two-fluid model of an axis-symmetric and non-rotating plasma plume, with the above mentioned characteristics, is closed by the following equations,

$$n_i = n_e = n \tag{2.1}$$

$$\nabla \cdot (n\boldsymbol{u_i} = 0) \tag{2.2}$$

$$\nabla \cdot (n\boldsymbol{u_e} = 0) \tag{2.3}$$

$$nm_i(\boldsymbol{u_i} \cdot \nabla)\boldsymbol{u_i} = -en\nabla\phi \tag{2.4}$$

$$0 = -\nabla \cdot \mathcal{P}_e + en\nabla\phi \tag{2.5}$$

$$0 = -\nabla \cdot \mathcal{P}_e + en\nabla\phi \tag{2.6}$$

$$u_{\theta i} = u_{\theta e} = 0, \tag{2.7}$$

with further assumptions such as:

$$m_e u_e^2, T_i \ll T_e \ll m_i u_i^2 \tag{2.8}$$

$$\nabla \cdot \mathcal{P}_e = \nabla p_e \tag{2.9}$$

The condition of quasy-neutrality in the plasma is stated by means of equation (2.1). This equation is used in lieu of more complex expressions, such as the Poisson Equation in plasmas

$$\nabla \cdot \phi = \frac{e}{\epsilon}(n_i - n_e),$$

in particular for plasma length scales larger than the Debye length ($\approx 1mm$ or less) [5]. In the far region plume expansion, the characteristic length of the problem is in the order of the thruster radii ($\approx 10cm$) and therefore, the equation of quasy-neutrality remains valid. Moreover, equations (2.2) and (2.3) express the conservation of mass for ion and electron species whereas equations (2.4) to (2.6) present the conservation of momentum for the same two species, and (2.7) indicates the presence of non-rotating ions nor electrons in the expansion. Lastly, the two assumptions (2.8) and (2.9) neglect in first instance the effects of ion thermal pressure and electron inertia (due to the typical scaling of the plasma plume in the far-region domain), and approximate the electron

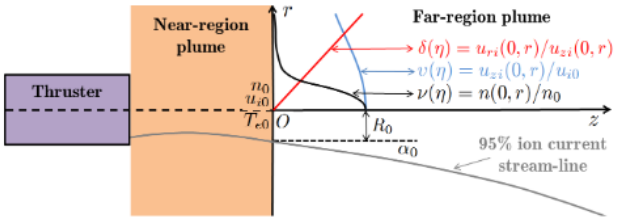


Figure 2.1: General description of a plasma plume. The initialization far-region plasma parameters ν, ϵ, δ are defined at the reference plane O, where the plasma streamtube is characterised with $r = R_0$ and the initial plume divergence angle α_0 [6]

pressure tensor $\mathcal{P}_e$ to a diagonal isotropic tensor.With these premises the complexity of the problem is greatly reduced, whereas kinetic theory would be needed for a complete definition of the problem. If this model of $\mathcal{P}_e$ is introduced in equation (2.5), together with the definition of a polytropic law for the electron cooling model, i.e.

$$T_e \propto n^{\gamma-1}, \tag{2.10}$$

the plasma potential ϕ reduces to:

$$\frac{e\phi}{T_{e0}} = \begin{cases} ln(n/n_0), & \text{if } \gamma = 1, \\ [(n/n_0)^{\gamma-1}]\frac{\gamma}{\gamma-1}, & \text{if } \gamma \neq 1, \end{cases} \tag{2.11}$$

where the effective plasma cooling rate is depicted by γ in (2.11). Values of the cooling rate are usually ranged from $\gamma = 1$ (the isothermal plume expansion lower limit) to $\gamma = 5/3$ (the adiabatic expansion upper limit) to match experimental results. Generally, the behavior of the far plume expansion is expected to be nearly isothermal, with $\gamma = 1 - 1.3$, as backed by [9], [10], [11] and [16].

In addition, combining (2.4) and (2.5), the following expression for the plasma momentum equation is obtained:

$$m_i(\boldsymbol{u_i} \cdot \nabla)\boldsymbol{u_i} = -\gamma T_e \nabla \ln(n) \tag{2.12}$$

At this point in the model derivation, it is important to recall that in the case of a non-magnetic plasma plume,the differential velocity between ions and electrons is expected to be small is small since the plume is globally current-free. Therefore, in first approximation, the assumption of $\boldsymbol{u_e} \simeq \boldsymbol{u_i}$ holds, and equations 2.3 can be dropped off.

The task consists firstly on how to solve for either the plasma density n or the ion velocity u_i properties, which are intrinsically coupled in the model, as seen in equations (2.2) and (2.12). In cylindrical coordinates these same equations can be rewritten as

$$u_{zi}\frac{\partial ln(n)}{\partial z} + u_{ri}\frac{\partial ln(n)}{\partial r} + \frac{\partial u_{zi}}{\partial z} + \frac{1}{r}\frac{\partial ln(ru_{ri})}{\partial r} = 0 \tag{2.13}$$

$$u_{zi}\frac{\partial u_{zi}}{\partial z} + u_{ri}\frac{\partial u_{zi}}{\partial r} = -u_{i0}^2\frac{(n/n_0)^{\gamma-1}}{M_0^2}\frac{\partial \ln(n)}{\partial z} \tag{2.14}$$

$$u_{zi}\frac{\partial u_{ri}}{\partial z} + u_{ri}\frac{\partial u_{ri}}{\partial r} = -u_{i0}^2\frac{(n/n_0)^{\gamma-1}}{M_0^2}\frac{\partial \ln(n)}{\partial z}, \tag{2.15}$$

where $M_0^2 = m_i u_{i0}^2/(\gamma T_{e0})$ is the square of the initial ion Mach number at $z = r = 0$, introduced in the previous equations as a non-dimensional parameter for its importance in future derivations. Particularly, within the hypersonic plume expansion limit $M_0 \gg 1$, certain approximations to be explained in subsequent chapters can be made, reducing the complexity of the previous model equations.

Finally, to conclude with the model, the initial profiles (at $z = 0$) of plasma density and ion velocity are needed as initial conditions. In particular,

$$n(0, r) = \nu(\eta) \qquad u_{zi}(0, r) = \upsilon(\eta) \qquad \frac{u_{ri}(0, r)}{u_{zi}(0, r)} = \delta(\eta) \tag{2.16}$$

where the new variable η represents the plume radius r at the initial plane, and δ is a measure of the plume divergence. The initial density and velocity fronts are typically measured in laboratory experiments, or otherwise estimated by complex near-region plume models, and their computation is beyond the scope of this dissertation. Hence, the far-region plume initial conditions will be considered to be known, and will be input directly into the previous model. With all this information, the resulting hyperbolic plasma problem can be solved following numerous approaches, but dissertation will focus in just three of them.

Chapter 3

The Asymptotic Expansion Method

In many cases, approximate solutions of plasma physical models are obtained by means of analytical techniques derived from Perturbation Theory. Perturbation methods rely on the presence of a small, and generally dimensionless parameter $\epsilon \ll 1$ in the problem. Indeed, equations (2.14)-(2.15) introduce the non-dimensional parameter $\epsilon = 1/M_0^2$, which clearly conditions the solution on the plume expansion model. In hypersonic plume expansions with $M_0^2 \gg 1$, and the model differential equations can be then solved under asymptotic expansions over the parameter ϵ, which is clearly small.
Particularly, the far-region plume expansion model within the physical assumptions formulated in previous chapters, can be sub-categorized into the regular perturbation level. In such problems, the unperturbed solution for non-zero small values of ϵ is physically similar to the unperturbed simpler solution when $\epsilon = 0$. Thus, the solution of the model can be generally expressed as a convergent expansion with respect to ϵ, consisting on the unperturbed case and higher order corrections [18].

3.1 The cold plasma solution

In the case of a fully hypersonic plasma, that is, pushing M_0 to an infinite value in equations (2.14) and (2.15), the global effect of the pressure is neglected and the plasma is said to be cold. In this limit or zeroth order solution, the unperturbed simpler solution of the model is recovered, with $\epsilon = 0$. Hence, the cold plasma density $n^{(0)}$ and ion velocity $u_i^{(0)}$ are not coupled in the model equations anymore, depending just on the initial front profiles ν, υ and δ.
From equations (2.14) and (2.15), u_i^0 is observed to remain constant along streamlines of the form

$$r = \eta + \delta z, \tag{3.1}$$

where $\delta \geq 0$ strictly, to avoid streamline crossing.
Therefore, the determination of an implicit $\eta = \eta(z, r)$ map from the above equation, leads to the direct integration of $u_{zi}^0(z, r)$, and $u_{ri}^0(z, r)$ from the initial plasma plume profiles. To that goal, a more suitable reference system is chosen to pass from the (r, z) map towards the (η, ζ) coordinates, where simply $\zeta = z$. By differentiating (3.1) in this new coordinates, the Jacobian matrix J for the map transformation is found:

$$J = \begin{bmatrix} \partial z/\partial\zeta & \partial r/\partial\zeta \\ \partial z/\partial\eta & \partial r/\partial\eta \end{bmatrix} = \begin{bmatrix} 1 & \delta \\ 0 & 1 + \zeta\delta' \end{bmatrix} \tag{3.2}$$

Now, if equation (2.14) is expressed in the (η, ζ) coordinates, the cold beam plume density takes the form

$$n^{(0)} = \frac{\nu}{(1 + \zeta\delta/\eta)(1 + \zeta\delta')}, \tag{3.3}$$

and The zeroth order solution is then closed with $u_{zi}^{(0)} = \upsilon$ and $u_{ri}^{(0)} = \upsilon\delta$. Although simple, the cold plasma limit solution gives a fast overall picture of the plume expansion in the far region, in this case in the form of a cone, without divergence angle growth.

3.2 First Order Correction

As previously stated, more accurate results can be obtained over the cold plasma limit in the AEM, by expanding the variables in the parameter $\epsilon = 1/M_0^2 \ll 1$:

$$\begin{aligned} u_{zi} &= u_{zi}^{(0)} + \epsilon u_{zi}^{(1)} + \epsilon^2 u_{zi}^{(2)} + ..., \\ u_{ri} &= u_{ri}^{(0)} + \epsilon u_{ri}^{(1)} + \epsilon^2 u_{ri}^{(2)} + ..., \\ lnn &= lnn^{(0)} + \epsilon lnn^{(1)} + \epsilon^2 lnn^{(1)} + ... \end{aligned} \tag{3.4}$$

The benefit of this expansion is the straightforward integration of higher order solutions, along the previously calculated zeroth-order streamlines, since the momentum and continuity remain decoupled. Thus, introducing the ϵ expansion into equations (2.14) and (2.15), the first-order velocity $u_{zi}^{(1)}$ is numerically solved by:

$$\begin{aligned} \upsilon\frac{\partial u_{zi}^{(1)}}{\partial\zeta} + \frac{\upsilon'}{1+\zeta\delta'}(u_{ri}^{(1)} - u_{zi}^{(1)}\delta) &= -u_{i0}^2\left(\frac{n}{n_0}\right)^{\gamma-1}\frac{\partial \ln n^{(0)}}{\partial z}, \\ \upsilon\frac{\partial u_{ri}^{(1)}}{\partial\zeta} + \frac{(\upsilon\delta)'}{1+\zeta\delta'}(u_{ri}^{(1)} - u_{zi}^{(1)}\delta) &= -u_{i0}^2\left(\frac{n}{n_0}\right)^{\gamma-1}\frac{\partial \ln n^{(0)}}{\partial r}, \end{aligned} \tag{3.5}$$

Once the ion plasma velocity is known, the plasma density is given by the following form of equation (2.13):

$$\upsilon\frac{\partial lnn^{(1)}}{\partial\zeta} = -u_{zi}^{(1)}\frac{\partial lnn^{(0)}}{\partial z} - u_{ri}^{(1)}\frac{\partial lnn^{(0)}}{\partial r} - \frac{\partial u_{zi}^{(1)}}{\partial z} - \frac{1}{r}\frac{\partial (ru_{ri})^{(1)}}{\partial r} \tag{3.6}$$

3.3 Higher Order Corrections

Again, the same procedure can be used to obtain higher order corrections to the model. Just for clarification, the Second Order Correction equations are offered explicitly below, as integrated in the plume numerical ode presented in this dissertation. The momentum and continuity equation of the Second Order AEM model at order ϵ^2 are reduced to,

$$\begin{aligned} \upsilon\frac{\partial u_{zi}^{(2)}}{\partial\zeta} + \frac{\upsilon'}{1+\zeta\delta'}(u_{ri}^{(2)} - u_{zi}^{(2)}\delta) &= -u_{zi}^{(1)}\frac{\partial u_{zi}^{(1)}}{\partial\zeta} - u_{ri}^{(1)}\frac{\partial u_{zi}^{(1)}}{\partial r} \\ &\quad - u_{i0}^2(\frac{n}{n_0})^{\gamma-1}[(\gamma-1)lnn^{(1)}\frac{\partial lnn^{(0)}}{\partial z} + \frac{\partial lnn^{(1)}}{\partial z}]), \end{aligned} \tag{3.7}$$

$$\begin{aligned} \upsilon\frac{\partial u_{zi}^{(2)}}{\partial\zeta} + \frac{\upsilon'}{1+\zeta\delta'}(u_{ri}^{(2)} - u_{zi}^{(2)}\delta) &= -u_{zi}^{(1)}\frac{\partial u_{ri}^{(1)}}{\partial\zeta} - u_{ri}^{(1)}\frac{\partial u_{ri}^{(1)}}{\partial r} \\ &\quad - u_{i0}^2(\frac{n}{n_0})^{\gamma-1}[(\gamma-1)lnn^{(1)}\frac{\partial lnn^{(0)}}{\partial r} + \frac{\partial lnn^{(1)}}{\partial r}]), \end{aligned} \tag{3.8}$$

,whereas the density equation (2.13) takes the following form:

$$\begin{aligned} \upsilon\frac{\partial lnn^{(2)}}{\partial\zeta} &= -u_{zi}^{(2)}\frac{\partial lnn^{(0)}}{\partial z} - u_{ri}^{(2)}\frac{\partial lnn^{(0)}}{\partial r} \\ &\quad - \frac{\partial u_{zi}^{(2)}}{\partial z} - \frac{1}{r}\frac{\partial (ru_{ri})^{(2)}}{\partial r} - u_{zi}^{(1)}\frac{\partial lnn^{(1)}}{\partial z} - u_{ri}^{(1)}\frac{\partial lnn^{(1)}}{\partial r} \end{aligned} \tag{3.9}$$

For presentation purposes, the more general velocity ith-correction expressions in the AEM model are depicted here as:

$$\upsilon\frac{\partial u^i_{zi}}{\partial\zeta}+\frac{\upsilon^{'}}{1+\delta^{'}\zeta}(u^i_{ri}-u^i_{zi}\delta)=-\sum_{j=1}^{i-1}(u^j_{zi}\frac{\partial u^{i-j}_{zi}}{\partial z}+u^j_{ri}\frac{\partial u^{i-j}_{zi}}{\partial r})-\sum_{j=1}^{i}T_e(\tilde{n}^{\gamma-1}\epsilon^{j-1})\frac{\partial ln\tilde{n}^{i-j}}{\partial z}) \quad (3.10)$$

$$\upsilon\frac{\partial u^i_{ri}}{\partial\zeta}+\frac{(\upsilon\delta)^{'}}{1+\delta^{'}\zeta}(u^i_{ri}-u^i_{zi}\delta)=-\sum_{j=1}^{i-1}(u^j_{zi}\frac{\partial u^{i-j}_{ri}}{\partial z}+u^j_{ri}\frac{\partial u^{i-j}_{ri}}{\partial r})-\sum_{j=1}^{i}T_e(\tilde{n}^{\gamma-1}\epsilon^{j-1})\frac{\partial ln\tilde{n}^{i-j}}{\partial r}) \quad (3.11)$$

In the previous two expressions, the term $T_e(\tilde{n}^{\gamma-1}\epsilon^{j-1})$ represents the coefficient of the ϵ^{j-1} in the Taylor Expansion series of $\tilde{n}^{\gamma-1}$ in ϵ on which the AEM corrections are founded upon, expressed as:

$$\tilde{n}^{\gamma-1}=(\tilde{n}^0)^{\gamma-1}\left(1+(\gamma-1)\epsilon ln\tilde{n}^1+\frac{(\gamma-1)^2}{2}\epsilon^2 ln^2\tilde{n}^1...\right)\cdot\left(1+(\gamma-1)\epsilon^2 ln\tilde{n}^2+\frac{(\gamma-1)^2}{2}\epsilon^4 ln^2\tilde{n}^2...\right) \quad (3.12)$$

Finally, the ith-order correction to the density is expressed as:

$$\upsilon\frac{\partial ln\tilde{n}^i}{\partial\zeta}=-\sum_{j=1}^{i}\left(u^j_{zi}\frac{\partial ln\tilde{n}^{i-j}}{\partial z}+u^j_{ri}\frac{\partial ln\tilde{n}^{i-j}}{\partial r}\right)-\frac{\partial u^i_{zi}}{\partial z}-\frac{1}{r}\frac{\partial ru^i_{ri}}{\partial r} \quad (3.13)$$

In is important to mention that every new order correction stands upon all previous order corrections and therefore, even though the first and second order AEM correction equations might seem cumbersome at first glance, the value of all the derivatives involved are either analytically known from previous corrections, or can be otherwise calculated by differentiating known variables along η. Nonetheless, this exact non-linear correlation of both velocity and density introduced by the ion convective and $(n/N_0)^{\gamma-1}$ terms in the previous equations, makes the model less traceable and more troublesome if further corrections are added. For simplification, the numerical method developed in this dissertation to solve the AEM model integrates up to the Second Order. It must be noted that these corrections are proportional to ϵ raised to the order power, and since $\epsilon=\frac{1}{M_0^2}\ll 1$ for supersonic and hypersonic plasmas, there exists a relation between the increase in accuracy and complexity in the AEM model when adding higher order corrections.

3.4 AEM Marching scheme

Probably, the most notable disadvantages of the Asymptotic Expansion Method are its nonuniform convergence and the loss of parameter accuracy downstream of the plume. These model deficiencies can be solved by introducing higher order corrections in the AEM equations (3.10)-(3.13). However, as concluded in previous sections, the introduction of such higher order corrections increases remarkably the complexity of the model and alternative approaches must be sought to increase the efficiency of the Asymptotic Expansion Method. One of those possible solutions consists in introducing a AEM reinitialization or marching scheme. This marching scheme advances (marches) front by front, recalculating and updating the output plume variables n, u_z, u_r,etc based on previous integration step results rather than expanding the solution downstream over the initially provided n_0, u_{z0}, u_{r0} fronts. In this way, the integration is sectioned in various steps delimiting axial domains along the plume. This reinitialization technique avoids the need to introduce higher order corrections in the model to increase the accuracy of the method, which might be otherwise very poor, even within the convergence region of the AEM (Please refer to **??** for a more detailed and visual comparison on the accuracy gain obtained by introducing the AEM marching method).

Essentially, in this reinitialization version of the AEM, the last n, u_r, u_z plume fronts calculated in a previous plume domain (integration step), are used as initial fronts for the next plume region integration. Hence, the reinitialization technique propagates the plume properties within a narrower axial region of interest avoiding numerical error growth and results which are out of the convergence zone. Conceptually, at defined axial locations in the plume where as $z^j = const$:

$$n^j=n^j(r) \qquad u^j_z=u^j_z(r) \qquad u^j_r=u^j_z(r), \quad (3.14)$$

where j represents the number of axial fronts or integration steps.Therefore,

$$\upsilon^j(r) = n^{j-1}(r) \qquad \nu^j(r) = u_z^{j-1}(r) \qquad \delta^j(r) = u_r^{j-1}(r)/u_z^{j-1}(r) \tag{3.15}$$

Again, by introducing this marching reinitialization procedure in the main AEM plume solver routine, the convergence region of the model is increased and less higher order corrections are needed to guarantee superior performance. In the end, the AEM marching method provides a more accurate radial and axial characterization of the far-region plume expansion, especially at longer downstream distances, that would otherwise be inexactly described with the simple AEM approach.

Chapter 4

The Self-Similar Method

One of the simpler, but not less accurate, approaches to obtain a rapid solution and quick physical understanding of the plume expansion is the Self-Similar Method. The SSM suggests modeling the plume expansion as a self-similar event, based on experimental measurements that prove the smooth and bell-shaped radial expansion of the plasma plume in the far region.
The SSM model imposes a couple of constraints into the problem, which are clearly deduced if the model is expressed in dimensionless form, by normalising the quantities with the values at $(z, r) = (0, 0)$ and a characteristic length R_0. In this way, the plasma properties are redefined as

$$\begin{aligned} \tilde{n} = n/n_0; \qquad \tilde{u}_{zi} = u_{zi}/u_{i0}; \qquad \tilde{u}_{ri} = u_{ri}/u_{i0}; \qquad \tilde{T_e} = T_e/T_{e0}; \\ \tilde{\phi} = e\phi/T_{e0}; \qquad \tilde{R} = R/R_0; \qquad \tilde{z} = z/R_0; \qquad \tilde{r} = r/R_0, \end{aligned} \tag{4.1}$$

together with the initial front conditions previously defined in (2.16):

$$\nu(\eta) = \tilde{n}(0, \tilde{r}); \qquad \upsilon(\eta) = \tilde{u}_{zi}(0, \tilde{r}); \qquad \delta\eta = \tilde{u}_{ri}(o, \tilde{r})/\tilde{u}_{zi}(0, \tilde{r}); \tag{4.2}$$

Observe that $\eta = \tilde{r}(0)$ is now the normalised radius at the initial form.
Within the assumption of self-similarity, the plasma streamlines expand radially with $\eta = const$ and

$$\tilde{r}(\zeta, \eta) = \eta h(\zeta) \tag{4.3}$$

This method introduces several scaling factors, in the case of equation (4.3) the self-similarity dilation function $h(\zeta)$ is used as a dimensioning parameter in our model. Moreover, in track with the SSM constraints, the plasma density and velocity can be propagated as mere functions of the initial plasma profiles ν and υ:

$$\begin{aligned} \tilde{n} &= \nu(\eta)\tilde{n}_a(\zeta) \\ \tilde{u}_{zi} &= \upsilon(\eta)\tilde{u}_a(\zeta) \end{aligned} \tag{4.4}$$

Again, the scaling functions $n_a(\zeta)$ and $u_a(\zeta)$ describe the evolution $\tilde{n}$ and $\tilde{u_{zi}}$ along the axis where $(\tilde{z}, \tilde{r}) = (0, 0)$.
If equation (4.3)is further derived in time, a simple relation between the radial and axial plasma velocities is found:

$$\tilde{u}_{ri} = \tilde{u}_{zi}\eta h^{'}(\zeta) \tag{4.5}$$

The above equation presents the first constraint in our model since at $\zeta = 0$, $\delta = \tilde{u}_{ri}/\tilde{u}_{zi} = C\eta$. Therefore, to find valid SSM solution, the initial plasma velocity profiles must be conical with

$$\delta^{'} = const \tag{4.6}$$

Now, if equation (4.4) is introduced into the plasma continuity and momentum equations (2.13) and (2.14) the resulting system relating u_a, n_a and $h(\zeta)$ is the following:

$$h^2\tilde{n}_a\tilde{u}_a = 1 \tag{4.7}$$

$$\upsilon^2 = -\frac{\nu^{\gamma-2}\nu^{'}}{\eta C} \tag{4.8}$$

$$M_0^2 \frac{h\tilde{u_a}(\tilde{u}_a h^{'})^{'}}{\tilde{n}_a^{\gamma-1}} = C, \tag{4.9}$$

where C is the last separation constant in the model. Equation (4.8) does indeed impose the second constraint between ν and υ in the SSM problem. In addition, if $\eta \to 0$ in this equation, the value of $C = -\nu^{''}(0)$ is obtained for consistency.
Besides (4.7) and (4.9), one more extra equation is needed to solve for the three unknown variables h, u_a and n_a. One could turn to the axial momentum equation (2.14) which in dimensionless SSM form is expressed as:

$$(\tilde{u}_a^2)^{'} - \tilde{n}_a^{\gamma-2}\tilde{n}_a^{'}\frac{2C\eta^2}{M_0^2}(\frac{nu}{nu^{'}\eta} - \frac{h^{'}\tilde{n}_a}{h\tilde{n}_a^{'}}) = 0 \tag{4.10}$$

However equation (4.10) is impractical, since it is not separable in ζ, η and leads to incongruent SSM solutions due to the impossibility to make the second term independent of η.Nonetheless, this equation can still be used as a measure of the differential error present in the SSM solution.
Therefore, an alternative expression to account for equation (4.10) is needed. In this dissertation, the option of $u_a = 1$ is chosen, although other choices exist and may render better results.

4.1 SSM Methods with $\tilde{u}_a = 1$

If relative variations in axial velocity are assumed to be of the order $O(1/M_0^2)$, the assumption $u_a = const = 1$ is valid and the SSM error is in this case proportional to $\frac{1}{M_0^2}$. Within this estimation, equation (4.7) reduces to:

$$\tilde{n}_a = 1/h^2, \tag{4.11}$$

while h is found by direct integration of equation (4.9),

$$(h^{'})^2 - (h^{'}(0))^2 = \frac{C}{M_0^2} \times \begin{cases} -(h^{2-2\gamma} - 1)/(\gamma - 1) & \text{if } \gamma > 1, \\ 2lnh & \text{if } \gamma = 1 \end{cases} \tag{4.12}$$

Using the transformation $h^{''} = h^{'}dh^{'}/dh$ and with $h^{'}(0) = \delta(0)$ for consistency. Finally the residual SSM error can be expressed by using equation (4.10) as:

$$\epsilon_l = \frac{C}{M_0^2}\frac{h^{'}}{h^{2\gamma-1}}\left(4\eta\frac{\nu}{\nu^{'}} + 2\eta^2\right) \tag{4.13}$$

Due to the physical constraints introduced by the model, the only degrees of freedom in the SSM solution are the Mach number M_0, the thermodynamical expansion constant γ, the value of $h^{'}(0)$ in equation (4.12) (or equivalently, the initial divergence of the plume $\delta(0)$, both related by $\tan\delta(0) = h^{'}(0)$) and finally, the appropriate initial density profile ν.
These constraints led several authors such as Parks, Ashkenazy and Korsun to develop specific solutions to the SSM, which are particularizations of the complete model developed in this dissertation.

Parks developed the model based on a isothermal plume with a Gaussian initial density profile and uniform axial velocity,

$$\gamma = 1; \qquad \nu = \exp(-C\eta^2/2); \qquad \upsilon = 1 \tag{4.14}$$

, whereas Korsun proposed a plume (isothermal or polytropic/adiabatic) with

$$\nu = \left(1 + C\frac{\eta^2}{2}\right)^{-1}; \qquad \upsilon = \left(1 + C\frac{\eta^2}{2}\right)^{-1/2} \tag{4.15}$$

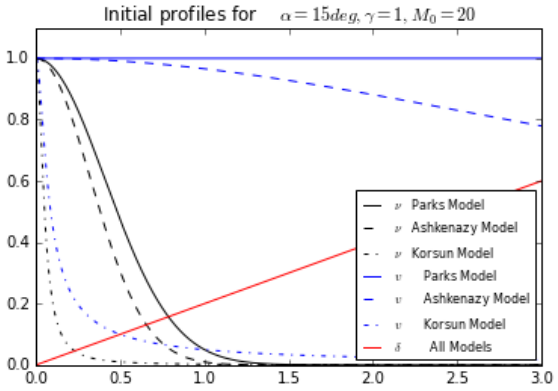


Figure 4.1: Initial density and axial velocity profiles considered by different authors.

Finally, Ashkenazy presented the following plume solution for the SSM:

$$\gamma = 1; \qquad \nu = \left(1 + k\eta^2\right)^{-C/(2k)}; \qquad v = \left(1 + k\eta^2\right)^{-1/2} \tag{4.16}$$

, where $k = (h^{'}(0))^2)$ to ensure a strict conical plume expansion.Figure 4.1 shows the initial far-region plasma plume profiles suggested by various authors, based on the previous derivations.

Chapter 5

Open-Source Code description and Implementation

The latest version of the code is described in this last chapter. The objective is to clearly summarize and document the overall structure of the code, as well as introduce its capabilities with respect to previous Plume Codes and its limitations.

5.1 General Concepts

The code has been written in Python code, an Open Source interpreted programming language well implemented in both the academic and professional fields. Characterized for its integrity and simplicity, Python has been used due to its vast set of predefined scientific libraries (Numpy,Scipy) which allow a high level of interaction with the programmer.
Moreover, the code has been developed following an Object Oriented Programming approach (OOP), where code re-usability is optimized and further functionalities can be added to the program by adding or editing existing code, instead of either changing existing code in-place or starting from scratch. In addition, the user is in this manner able to treat the Plume as an entity or object, with clearly defined attributes and methods which enhance plume data visualization and comparison. Additionally, the code presented in the dissertation has followed and benefited from a Test-Driven-Development (TDD) approach. TDD is commonly referred as the process of implementing code by writing tests on its structure and implementation first. The approach defines a characteristic cycle where the developer writes tests, observes them fail, and finally writes the code to make them pass.However, TDD is not just about mere code testing. By implementing TDD, the structure and design of the different functionalities is carefully thought before hand, leading to more testable, maintainable and readable code yet simple and elegant.

Thus, the AEM and SSM have been coded following these programming approaches in order to simulate the far region expansion of a plasma plume. The code has been adapted to integrate a solution for plasmas with the same initial far-field density and velocity profile distributions, at a characteristic initial potential electric field, temperature and thermodynamical expansion constant γ.

5.2 Plasma Plume Code Structure and Interface

The code structure presented in the HyperPlume project has been dissected and modularized into different subfolders, to make the most out of OOP and TDD techniques. The network of folders that have been sub-directed is schematically depicted in figures 5.2. In this manner, the semi-analytical model AEM and SSM codes is independent, and inherit common base methods from the parent class Hyperplume. This architecture of folders is integrated as a complete Python Plume Code package that facilitates relative import of libraries, methods, variables and eases the

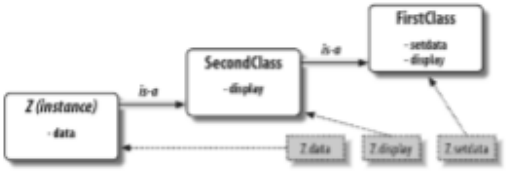


Figure 5.1: Simple tree diagram of OOP code that describes the structure and logic of the Plasma Plume code. Object Z is initialized as an instance of *SecondClass*, and inherits the class methods *setdata* and *display* from the parent metaclass *FirstClass*.However, *display* method has been customized in *SecondClass* as a particularization for its instances. Finally, the object Z, inherits the property *data* from the class.

comparison and visualization of AEM and SSM plumes. The testing suite is separated from the main code adding integrity to the plume package.

```
HyperPlume/

    ...

    src/
        __init__.py

        AEM/
            __init__.py
            AEM_plume.py

        SSM/
            __init__.py
            SSM_plume.py

        HYPERPLUME/
            __init__.py
            hyperplume.py

    tests/
        test_HYPERPLUME/
            test_HYPERPLUME.py
        test_AEM/
            test_AEM.py
        test_SSM/
            test_SSM.py

    doc/
        HyperPlume.html
        TFG.pdf
```

Figure 5.2: Structure of folders and files in HyperPlume package.

This folder structure clearly identifies the different plume scripts developed and included in the package. *AEM_plume.py, SSM_plume.py and hyperplume.py* are the three main Python scripts needed for the correct integration and simulation of AEM and SSM plumes.In particular, *hyperplume.py* includes the parent class **Hyperplume** which introduces common base methods to be inherited by the child classes **AEM** and **SSM**. These two classes are included in *AEM_plume.py, SSM_plume.py* scripts respectively, and perform the creation and simulation of specific plumes by particularizing the more general solution methods introduced in **Hyperplume**. In 5.1 a summarized description of the shared methods involved in successful plume creation is presented.

```
1
2  class AEM(Hyperplume):
3
4  """ Parent class Hyperplume loads target plasma and defines common attributes as
       well as
5      shared methods in the AEM and SSM plume classes"""
6
7      __metaclass__= abc.ABCMeta # Python decorator used to define abstract methods
       at any location in the class
8
9      @abc.abstractclassmethod # Defining abstract method
10
11     def solver(self):
12
13         """Solver Abstract Method to be particularised by each Plume code. It is
       only defined for
14         structure purposes in parent class Hyperplume"""
15
16         return
17
18     @abc.abstractclassmethod
19
20     def query(self,z,r):
21
22         """Query abstract method returns plasma profile data at specified grid
       points. query method is
23         to be particularised by each plume code.It is only defined for structure
       purposes
24         in parent class Hyperplume"""
25
26         return
27
28     def __init__(self,plasma={'Electrons': {'Gamma': 1,'T_0_electron': 2.1801714e
       -19,'q_electron': -1.6e-19},'Ions': {'mass_ion': 2.1801714e-25, 'q_ion': 1.6e
       -19}},z_span=np.linspace(0,100,500),r_span=np.linspace(0,40,500),n_init=0.0472*
       np.linspace(1,0,500)**2):
29
30         """ plume_constructor loads common class properties for AEM and SSM plume
       classes
31
32         INPUTS
33         ==========
34
35         * plasma ------> simple_plasma object dictionary
36
37         * z_span ------> axial region where the problem will be integrated
38
39         * r_span ------> radial region where point will be integrated
40
41         * n_init ------> initial dimensional density front
42         """
43
44         self.plasma = plasma
45         self.Gamma = plasma['Electrons']['Gamma']
46         self.T_0 = plasma['Electrons']['T_0_electron']
47         self.m_ion = plasma['Ions']['mass_ion']
48         self.q_ion = plasma['Ions']['q_ion']
49         self.z_span = z_span
50         self.eta = r_span
51         self.n0 = n_init
```

Listing 5.1: Hyperplume Class main methods

The previous code listing describes three important methods defined in **Hyperplume**: the plume `self.__init__` constructor method, the plume solver self.solver and finally a query method to obtain the characterization of the plume at specific downstream locations, `self.query`.These methods are included abstractly in the parent class for code structure and interface integrity, increasing readability and cleanness in the code code.Furthermore, these methods are inherited and particularized in **AEM** and **SSM** subclasses, attending to the specific integration needs and

equations of each plume expansion model. In 5.2, snippets of the particularization of such classes is depicted.

```
1
2  class AEM(Hyperplume):
3
4      """Class AEM inherits methods __init__,solver and query form parent class
       Hyperplume, and particularizes them.All initial
5      inputs must be given in dimensional form."""
6
7      """ Additional documentation and code not described in the current listing"""
8
9  def __init__(self,plasma={'Electrons': {'Gamma': 1,'T_0_electron': 2.1801714e-19,'
       q_electron': -1.6e-19},'Ions': {'mass_ion': 2.1801714e-25, 'q_ion': 1.6e-19}},
       z_span=np.linspace(0,100,500),r_span=np.linspace(0,40,500),n_init=np.linspace
       (1,100,500),uz_init=np.linspace(1,100,500),ur_init=np.linspace(0,100,500),
       sol_order=0):
10
11         """ Class method __init__ is used as class constructor. Inputs are passed
       and stored as attributes."""
12
13         #Call parent class Hyperplume constructor method to store main plasma
       properties as attributes in the AEM class."""
14         super(AEM, self).__init__(plasma,z_span,r_span,n_init)
15
16         self.uz0,self.ur0,self.d0 = uz_init,ur_init,ur_init/uz_init #Load
       additional AEM plasma plume properties
17         self.alpha0=math.degrees(math.atan(interp1d(self.eta,self.d0)(1))) #Initial
        Plume divergence at the 95% streamline
18         self.order = sol_order #AEM Solution Order
19         self.eps = self.Gamma*self.T_0/(self.m_ion*(uz_init[0]**2 + ur_init[0]**2))
       #residual expansion parameter. Inverse of squared Mach Number
20         self.M0 = np.sqrt(1/self.eps) #Plume Mach Nmeber
21
22         """ Derivatives of initial front"""
23
24         self.d0p = self.eta_deriver(self.eta,self.d0) #derivative of plume
       divergence
25         self.d0p[0],self.d0p[-1] = self.d0[1]/self.eta[1],self.d0p[-2] + (self.d0p
       [-2] - self.d0p[-3]) #Edge vetor prime conditions
26         self.uz0p = self.eta_deriver(self.eta,self.uz0) #derivative of initial
       axial velocity
27         self.uz0p[0],self.uz0p[-1] = 0,self.uz0p[-2] + (self.uz0p[-2] - self.uz0p
       [-3])
28         self.duz0p = self.eta_deriver(self.eta,self.d0*self.uz0) #derivatie of
       initial radial velocity
29         self.duz0p[0],self.duz0p[-1] = self.duz0p[1]/self.eta[1],self.duz0p[-2] + (
       self.duz0p[-2] - self.duz0p[-3])
30
31         """Grid Set Up"""
32
33         self.z_grid,self.r_grid = self.grid_setup(self.z_span.size,self.eta.size) #
       2D grids of z, and r points in the plume
34
35
36 class SSM(Hyperplume):
37
38     """Class SSM inherits methods __init__,solver and query form parent class
       Hyperplume, and particularizes them.Any initial density
39     profile is valid, and given in dimensional form."""
40
41     """ Additional documentation and code not described in the current listing"""
42
43     def query(self,z,r):
44
45         """ Method query returns the density, velocity profile, temperature, the
       electric potential and SSM error at
46         particular (z,r) points in the Plume."""
47
48         eta = r/self.h_interp(z) #calculation of eta at user targeted grid point
```

```
49
50          n = self.n0[0] * self.nu_interp(eta) * 1/self.h_interp(z)**2 #Dimensional
        density at targetd (z,r) points
51
52          #Calling various Hyperplume methods to calculate remaining plasma
        parameters based on plume density
53
54          T = self.temp(n,self.n0[0],self.T_0,self.Gamma) #Dimensional Temperature at
         targetd (z,r) points
55
56          phi = self.phi(n,self.n0[0],self.T_0,self.Gamma,self.q_ion) #Dimensional
        potential at targetd (z,r) points
57
58          u_z = self.M_0*np.sqrt(self.Gamma*self.T_0/self.m_ion) * self.
        upsilon_interp(eta) #Dimensional axial velocity at targetd (z,r) points
59
60          u_r = self.d_0 * u_z * self.dh_interp(z) * eta #Dimensional radial velocity
         at targetd (z,r) points
61
62          error = self.C * self.dh_interp(z) / (self.M_0**2 * (self.h_interp(z)**(2*
        self.Gamma-1))) * (4 * eta * self.nu_interp(eta) / self.nu_prime_interp(eta) +
        2 * eta**2) #SSM error at targetd (z,r) points
63
64          return n,u_z,u_r,T,phi,error,eta
```

Listing 5.2: AEM and SSM Class Code Particularization main methods

In the end, **Hyperplume** just establishes the fundamental programming skeleton and bottom functionalities in the plume code, while **AEM** and **SSM** perform the main integration routines.Obviously, there are additional methods in both the parent and child classes.These methods, their implementation and syntax care fully detailed in the documentation of the code accompanying this dissertation.
Focusing the attention now in the interaction of user and code, snippet code listing 5.3 shows the main steps in plume creation and characterization. As explained in earlier sections, the OOP approach allows the user for simple plume object creation, and to treat such plume object as an entity with defined properties and methods.

```
1
2 """ Simple Steps in Plume creation and expansion type characterization"""
3
4 #Calling Hyperplume Plasma Creation method self.simple_plasma(particle_charge,
      ion_mass,electron_temp,thermdynamicall_expansion_const)
5
6 Plasma = Hyperplume().simple_plasma(1.6e-19,2.1801714e-25,2.1801714e-19,1.2)
7
8 #Determination of initial far-field plume profiles
9
10 z_span = np.linspace(0,100,500) #Far-field plume axial domain
11 r0_span = np.linspace(0,3,500) # Initial far-field plume radial profile
12
13 n0 =  np.exp(-6.15/2*eta_0**2) #Initial far-field plume density
14
15 #For AEM class
16
17 uz0 = np.linspace(20000,20000,500) #Initial far-field axial plume velocity
18 ur0 = np.linspace(0,2.183821405597214e+04,500) #Initial far-field radial plume
      velocity
19 sol_order = 2 #Solution order (higher order correction) of results in AEM model
20
21 #For SSM class
22
23 d0=0.2679491924311227 #Initial far-field plume divergence ur0/uz0
24 M0 =20                 #Characteristic initial far-field plume Mach Number
25
26 """ Plume type creation """
27
28 PlumeAEM = AEM(Plasma,z_span,r0_span,n0,uz0,ur0,sol_order) #AEM plume
29 PlumeSSM = SSM(Plasma,M0,d0,z_span,r0_span,n0) #SSM plume
30
```

```
""" Plume characterization """

PlumeAEM.solver() #Solves AEM plume using model equations in the entrire grid
PlumeSSM.solver() #Solves SSM plume using model equations in the entire grid

z,r = np.linspace(85,85,50),np.linspace(0,50,50) #Section of plume grid.

lnn_AEM,uz_AEM,ur_AEM,T_AEM,phi_AEM,etaAEM = PlumeAEM.query(z,r) #Extraction of
    plume variables at specified grid points
n_SSM,uz_SSM,ur_SSM,T_SSM,phi_SSM,etaSSM = PlumeSSM.query(z,r) #Extraction of plume
     variables at specified grid points
```

Listing 5.3: User steps in Plume creation and characterization

5.3 Plasma Plume Code Documentation

The Python Plume code presented in this dissertation has been coded following the *PEP 8 - Style Guide Standards* for python scripting. In addition, the code has been supplemented with a battery of tests scripts that check the validity and integrity of the methods. To that end, the *Unittest test framework* module, a module inherent to Python testing, has been used to implement TDD approaches. As previously commented, Python is a high level programming language that uses modules,complete set of predefined libraries, which aid the programmer in developing front-end applications by providing common functions or methods. Besides the *Unittest test framework*, three other important core packages have been used in the plume code: the *NumPy* and *Matplotlib* modules, integrated in the SciPy Python ecosystem for mathematics science and engineering. *NumPy* is the foundational dimensional array module, used for scientific computation and as an efficient multi-dimensional container of generic data.On the other hand, *Matplotlib* is the basic 2D and 3D visualizer of Python data, used to generate all the figures represented in this dissertation.

On another note, a pillar of every code is its documentation, used as an user guide or otherwise needed for future development of the code. The documentation of the code has been advantageously achieved by means of *Sphinx*, a software tool that creates automatic documentation of Python code in a variety of formats, such as *PDF* or *HTML*. Figure 5.3 shows the main index page of HyperPlume's project documentation in *HTML* format. In addition, the complete documentation in *PDF* format of the HyperPlume project, including different classes, methods and user implementation is included in annex A.

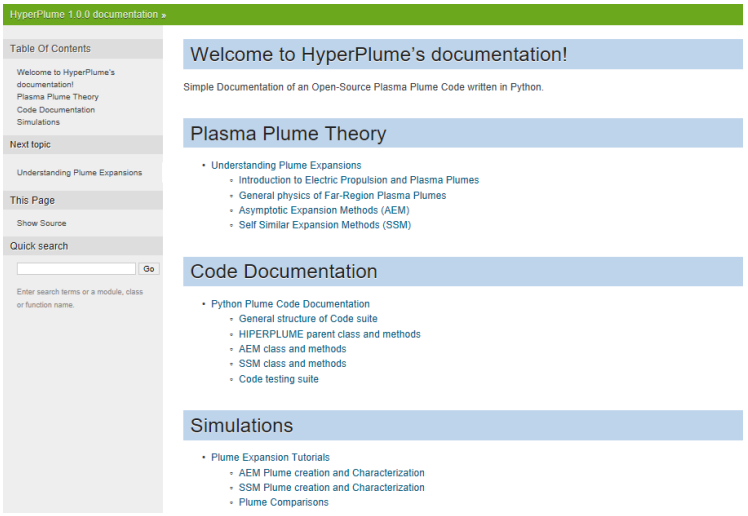


Figure 5.3: Index webpage of HyperPlume project Documentation

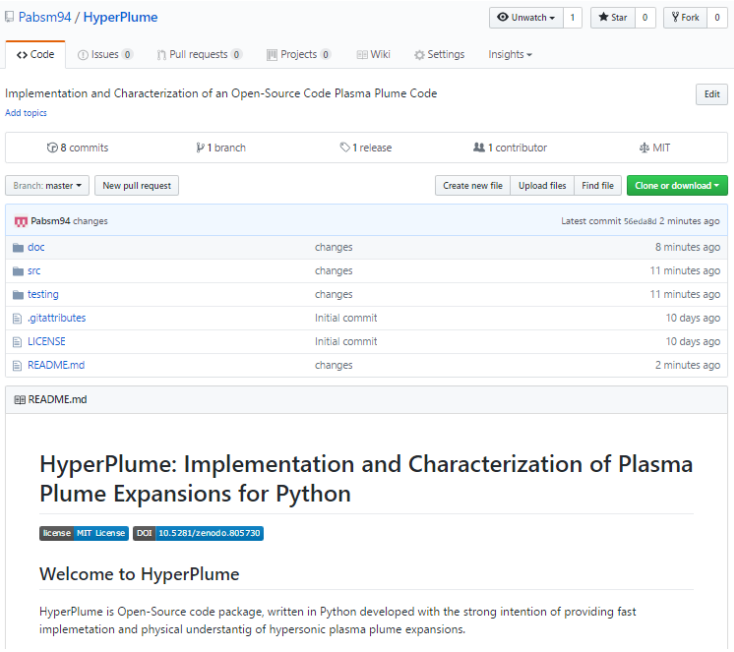


Figure 5.4: Structure of HyperPlume web repository in GitHub.[31]

5.3.1 Open-Source Code repository and licensing

Pursuing an open-source plume expansion software where developers and scholars can collaboratively integrate new methods and solutions, merge code, and interactively update the project, the plasma plume package developed in this dissertation has been uploaded by means of the web repository GitHub, under the project name *HyperPlume*. A repository is traditionally known as a particular location, accessed by a unique URL, where all the contents of a particular project are safely stored and versioned. GitHub is a web hub for repositories,allowing developers to create fully connected software networks and work collectively as a like-minded unity. GitHub is backed upon Git version control systems, which keep code revisions and modifications straight after the first official or “beta” release of software. Git is also the preferred version control system by developers, thanks to its decentralized accessibility and efficiency in storing files while ensuring file integrity.
Thus, the plasma plume software package encapsulating all the simulations and results to be presented in following pages, can be thought as the official ”beta” version release of an open source plume expansion code. To secure this first release and protect the authorship of the work, a license has been attached to the GitHub repository. In particular, the soft and permissive MIT license has been selected, granting the distribution of licensed works and code revisions under different terms and without the explicit “beta” source code. This short license only requires copyright preservation and license modification notices.
Finally, in order to make the plume expansion software citable in academic literature, a Digital Object Identifier (DOI) has been assigned to the GitHub repository with the data archiving tool Zenodo. A DOI is an individual alphanumeric string accredited by the International DOI Foundation to identify software and provide a persistent link to its location on the Internet even if this location and other metadata changes. In this manner, the plume expansion code can always be tracked and be appropriately cited and used among scientific community.Figure 5.4 shows the friendly user interface of the HyperPlume project on GitHub.

5.4 Differences with previous Far-Region Plasma Plume codes

The new Python code presented in this dissertation introduces an improvement in the integration process with respect to the previous Matlab version. Indeed, these improvements extend the spectrum of plume types which could be investigated and solve important glitches in the calculation of plasma properties.Precisely due to these changes, there exist numerical small numerical discrepancies in the results between the existing ant the new Python plume codes.
The code upgrades included in each method as described in the following sections.

5.4.1 AEM new functionalities and adjustments

One of the advancements in the new AEM Python Code is the introduction of the marching plume solver, a reinitialization method which increases the accuracy of the AEM general framework.
Additionally, When AEM 1^{st} and 2^{nd} density correction equations 3.6 and 3.9 are evaluated at the axis $r = 0$, a singular density solution cannot be reached due to the indetermination in the term

$$\lim_{r\to 0} \frac{1}{r} \frac{\partial (r u_{ri})^{(i)}}{\partial r} = \frac{0}{0} \tag{5.1}$$

If the derivative is expanded in the previous expression, then

$$\lim_{r\to 0} \left(\frac{u^i_{ri}}{r} + \frac{\partial u^i_{ri}}{\partial r} \right), \tag{5.2}$$

and the indetermination is just isolated to the first term $\frac{u^{(i)}_{ri}}{r}$. Applying L'Hopital rule to this first term, equation (5.1) can be simply substituted at the plume axis by

$$\lim_{r\to 0} \frac{1}{r} \frac{\partial (r u_{ri})^{(i)}}{\partial r} = 2 \frac{\partial u^{(i)}_{ri}}{\partial r}. \tag{5.3}$$

The new developed Python plume software introduces this exact axis correction in the code, while analogous far-region plume codes substitute this limit value by an approximated one very near the axis.
The change leads to high relative density error in the marginal zone near the axis between the two plume codes. However, this error reduces in few steps further from the axis, due to the introduction of the higher order corrections in plasma velocity u^i_{zi}, u^i_{ri} in equations (3.6) and (3.9).
Finally an exact query method has been introduced in the new python version of the AEM, where the user can read plasma parameters at specific locations in the grid, instead of obtaining an overall parameter map along the entire plume grid.

5.4.2 SSM new functionalities

As explained in section 4, the SSM imposes two constraints in the plume model. The user is only free to choose Although these plume choices are completely valid within the SSM derivations, the solutions obtained from these theoretical models are quite restrictive and do not match experimental data in most of the occasions. In previous far-region plume codes, the SSM solutions were just based in these models and therefore, the user was limited in the selection of plume types. Moreover, previous codes set analytically the density and axial velocity distributions ν and ϵ, disregarding the SSM constraint on these two parameters as expressed in equation (4.8) and leaving the user with the only choice of M_0 and δ.
The new Python code represents a more flexible fit for experimental data, where the user inputs ν directly as a vector, and the code follows the complete SSM derivations to obtain the bes solution. Thus, a greater accuracy for all types of plumes is reached in the regions of interest. Nonetheless, the python code incorporates the possibility of integrating such particular and theorized plumes in case the user wants a quick physical understanding of the expansion properties.

Chapter 6

Simulation and Results

In this section the characterization of plumes with interesting values of plume divergence, march number or simulating different expansion models is produced. Firstly, the results from the two semi-analytical models AEM and SSM are presented and commented on an individual basis

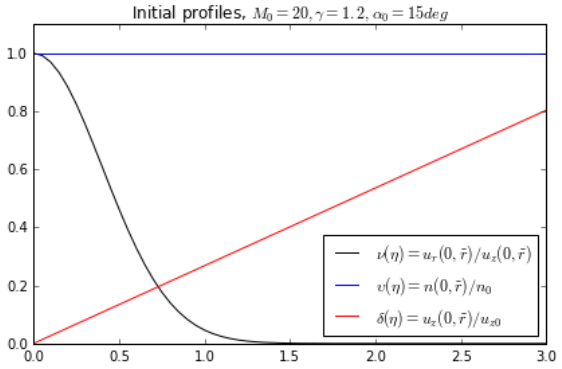


Figure 6.1: Initial normalized plasma profiles used in subsequent analysis, based on a Gaussian density profile, a uniform axial velocity field and a divergence for $\alpha_0 = 15deg$ and $\gamma = 1.2$

6.1 AEM characterization

So far in this dissertation, the Asymptotic Expansion Method has been presumed valid in any plume configuration. However, the simulations are only accurate when the series expansion of equation (3.4) converges in a finite region of interest. To accurately examine this convergence region important plasma variables must be tracked, and any perturbations of these variables along the plume must be also assessed. In the AEM case, such perturbations are expressed as the *i-th* order corrections of the plasma density and velocity profiles variables along the plume.In any case, n and u_z, u_r are not the only variables to identify in the analysis of plumes. The characterization of the ambipolar electric potential ϕ and the far field divergence angle α_{FR} evolution along the plume is also necessary in preliminary electric propulsion designs, and will be addressed in later sections of this dissertation (See chapter 7).
Visually, Figure 6.2 describes the dissimilarities between the AEM zeroth, first and second order corrections, at a unique axial location in the plume. From the plot, the greatest offset of the first order correction in plasma density with respect to the other solution orders can be grasped. As a matter of fact, this result can be preliminary used to infer the condition that determines the convergence region in the AEM, as will be explained hereafter.

Figure 6.3 plots (a,b) describe the evolution of plasma density $\ln(\tilde{n})$, for the zeroth,first and second order AEM corrections and specific values of plume initial Mach number, divergence angle values under the unique thermal expansion model determined by γ.As anticipated, plasma density

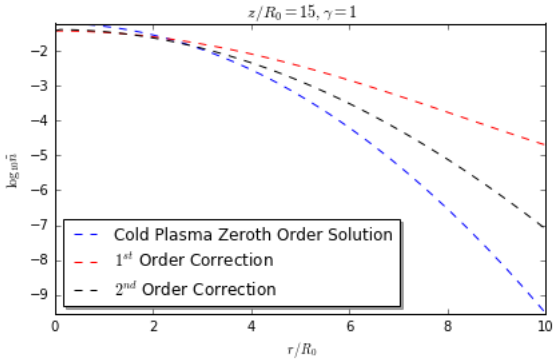


Figure 6.2: AEM plasma density values (logarithmic form) comparison for a plume with $M_0 = 20$ and $\alpha_0 = 0.2$ between the three AEM order corrections in the new developed Python Code.

decreases downstream of the plume and radially.Plasma density and velocity profiles do indeed depend on the chosen M_0, α_0 and γ, and parametric studies over these criteria facilitate the analysis of plumes. To that end,figure 6.3 plots (c,d) show the density and axial velocity perturbations for different values of initial divergence angle and γ. The plots indicate that the most critical correction is found in the density, increasing significantly faster than the velocity correction, reaching at $z = 100$ values as large as 900 for the $\eta = 1$ streamline in the isothermal $\gamma = 1$ case. It is found from the same plots that the perturbations grow faster and boundlessly for both lower α_0 and γ.In view of these results, the condition used to explore the convergence of the method can be expressed as $\epsilon \ln(\tilde{n})^1 < \ln(\tilde{n})^0$ in the density correction.This criterion is advantageous in both bounding the region of interest, and determining the plasma front where the simulation must be stopped and restarted before the perturbations grows excessively. This reinitialization procedure increases the accuracy of the AEM method and increases the extension of the convergence region further downstream.
Finally, figure 6.4 describes this validity region for the limiting plumes, in terms of density perturbation growth as depicted in Figure 6.3 plots (a,b), with $\gamma = 1$ and $\alpha_0 = 15$. As expected, the convergence region of the AEM model becomes wider axially and radially with higher M_0.

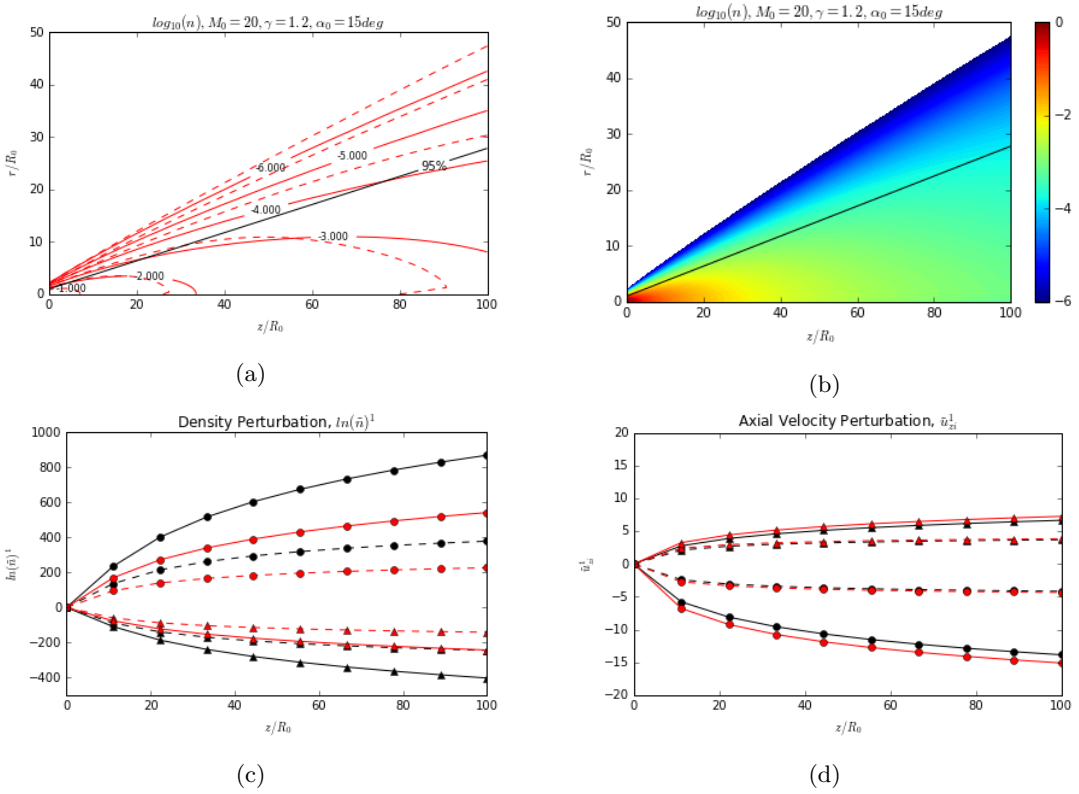


Figure 6.3: Plasma plume density logarithm contours (a) for the AEM zeroth (solid lines) and second order(dashed lines) corrections, as well as, plume density filled contour for the AEM first order correction (b) with 95% ion current streamline $\eta = 1$ (black solid line). First order perturbations in density logarithm (c) and axial velocity (d) for $\gamma = 1$ (solid lines) and $\gamma = 1.2$ (dashed lines), for $\alpha_0 = 15$ (black lines) and $\alpha_0 = 20$ (red lines) at gird axis (circles) and $\eta = 1$ streamline (triangles)

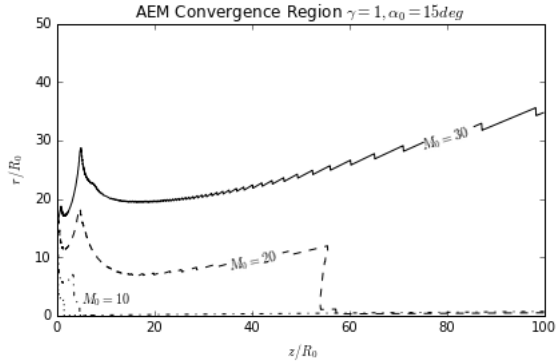


Figure 6.4: Convergence Region shown for several choices of plume Mach numbers.

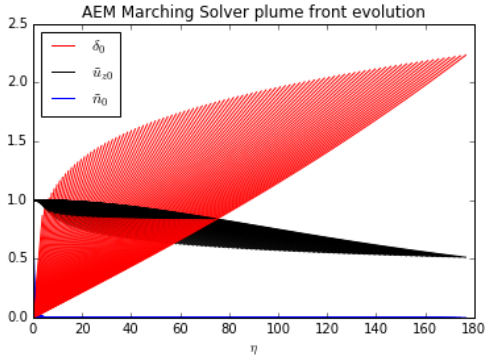


Figure 6.5: Evolution of re-initialized plume fronts $\upsilon^j(r), \nu^j(r)$ and $\delta^j(r)$ at each integration step in AEM marching solver method. In this case, the characterization of the plume was performed with a marching AEM scheme using 100 axial integration steps.

6.1.1 AEM Reinitialization Marching Scheme Results

As previously stated, further accuracy benefits can be achieved by introducing a marching scheme into the AEM numerical integration. Figure 6.5 describes the smooth evolution of the recalculated $\upsilon^j(r), \nu^j(r), \delta^j(r)$ fronts at each integration step along the plume grid, using the marching plume solver. The simple approach characterizes the plume expansion from fixed density, velocity and divergence profiles like the ones depicted in 4.1 or 6.1, and propagates these initial far-region plume conditions downstream by means of the model equations. This approach however, leads to high numerical error and results falling out of the convergence zone of the method. On the contrary, the marching scheme reinitializes $\upsilon^j(r), \nu^j(r), \delta^j(r)$ fronts at each axial integration step, limiting the error growth in the calculations of the plume expansion variables. In addition, figure 6.6 depicts the main results of such reinitialization technique, compared to baseline AEM figures. As expected, plume properties vary downstream of the plume where simple AEM error grows excessively. In addition, in the isothermal plume expansion limit expressed with $\gamma = 1$, greater downstream divergence between simple and marching scheme AEM outputs is found.Moreover, at lower values of M_0 and α_0 this difference accentuates, as observed in plots (c) and (d) plotting the relative error

$$\epsilon_{rel} = \frac{\log_{10}(n)_{AEM_{base}} - \log_{10}(n)_{AEM_{marching}}}{\log_{10}(n)_{AEM_{marching}}}.$$

Indeed, as stated derived in previous sections, the isothermal plume expansion sets the critical limit in terms of simple AEM results, even more when the plume Mach number is low, thereby expecting higher accuracy gains by introducing the AEM marching procedure in these cases.

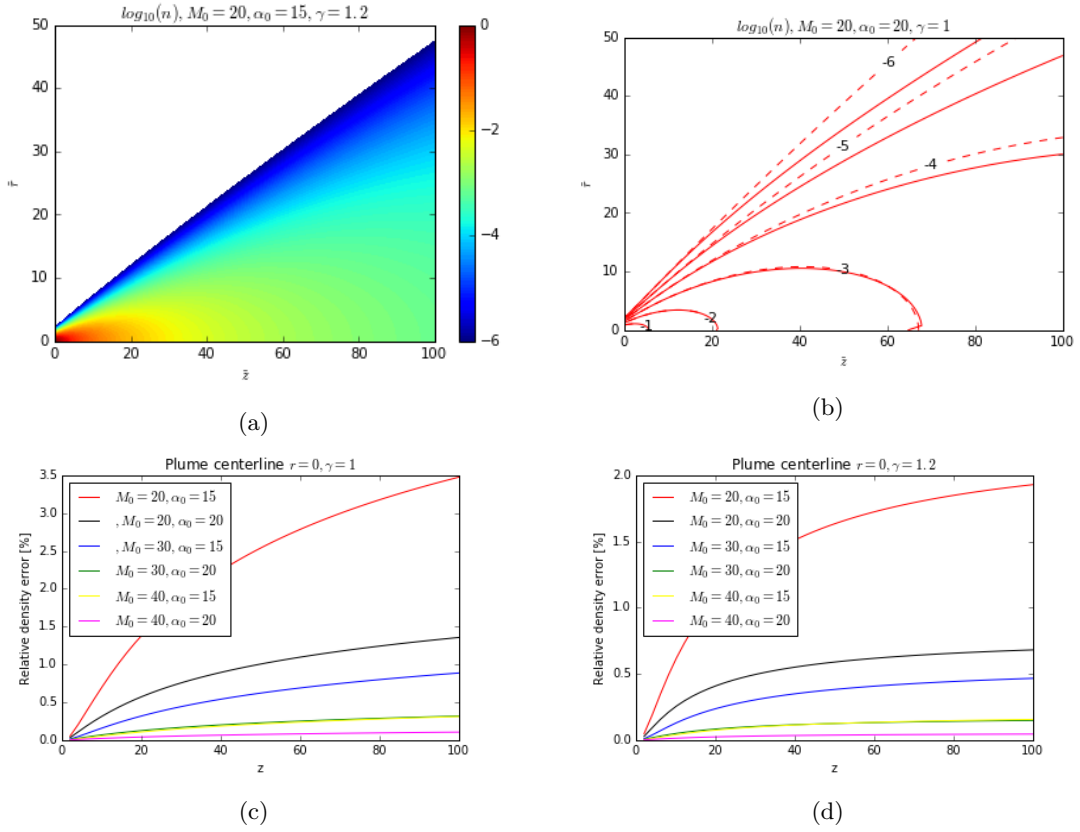
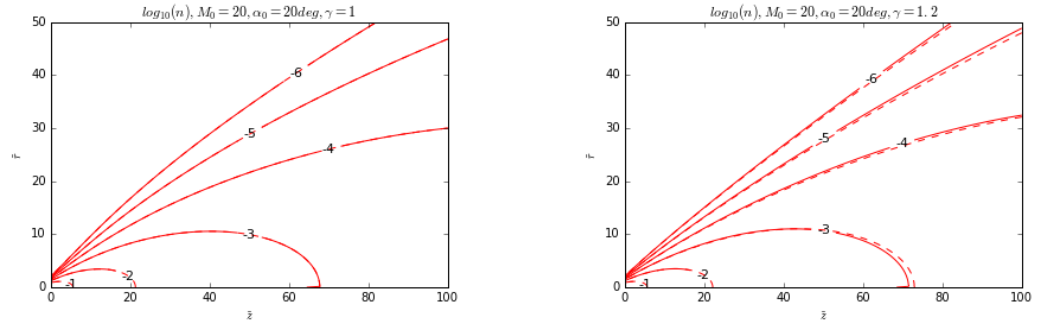


Figure 6.6: Plasma reinitialization scheme plume density contours, for the AEM second order correction (a) and comparison against AEM baseline second order correction results (b).Relative density error accuracy comparison between both methods along plume axis, for different values of M_0 and α_0 (c) and (d).

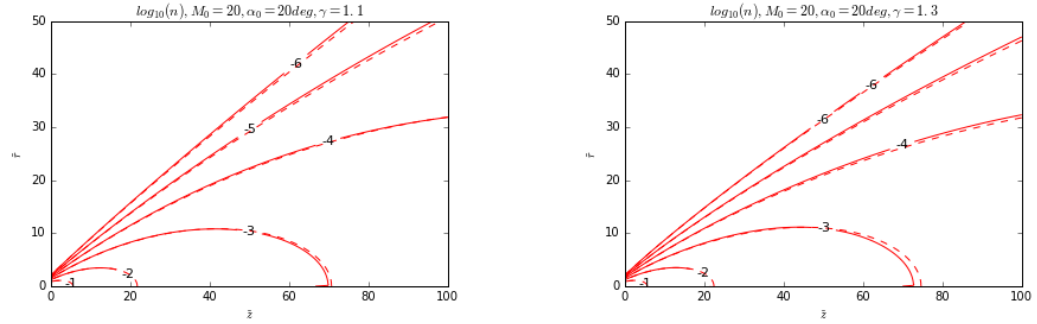
6.1.2 AEM Validation against previous Plume Expansion software

In order to substantiate the former results, several validation tests were performed against the Matlab code and results derived in 6.1. Plot (a) in Figure 6.7 shows the direct comparison between the AEM second order contour density values (logarithmic form) along the entire grid of a plume with unique M_0, α_0 values, and under different choices of γ. Exacts results are retrieved when comparing isothermal plume expansions as observed in plot(a). Divergence seems to appear only in the case of non-isothermal plume expansions, with increasing values of γ as seen in plots (b),(c),(d). The mismatch is especially quantifiable at moderate-to-large radial and axial distances downstream in the plume.
Although qualitatively small, this difference is quantified and inspected in figure 6.8, at unique $\tilde{r}$ and $\tilde{z}$ locations in the plume and under different γ values, for the AEM second order solution. The relative error ϵ_{code} depicted in plots (c) and (d) was calculated as

$$\epsilon_{code} = \frac{\ln(n)_{Python} - \ln(n)_{Matlab}}{\ln(n)_{Matlab}}.$$

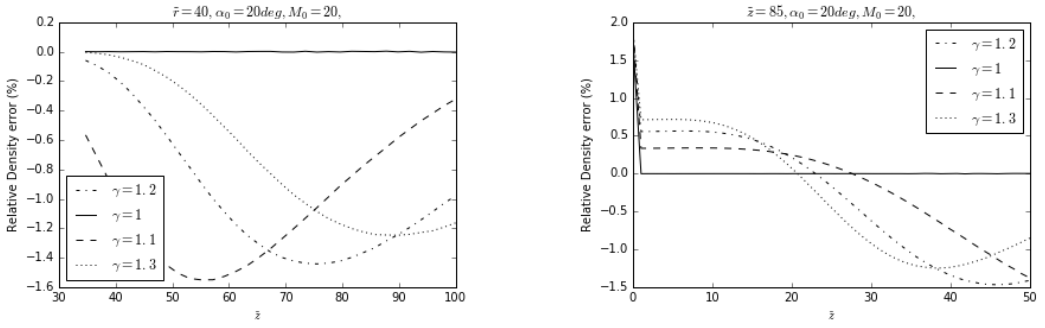


(a) (b)



(c) (d)

Figure 6.7: AEM plasma density error comparison for a plume with $M_0 = 20$ and $\delta_0 = 0.2$ between the new Python Code and previous Matlab models. AEM benchmark results have been used, with no reinitialization. Black dashed line represents raw data error, while smooth red line follows a best fit line of the results (a). Density relative error comparison near the axis, between new Python plume code and previous plume software, for an initial Gaussian density profile plume with the same cited parameters (b).



(a) (b)

Figure 6.8: Second order AEM Density error comparison between Python and Matlab codes, at unique radial and axial locations in the plume under different values of the thermal expansion constant γ.

The exact same behavior was observed when analyzing AEM first order correction results but on the other hand, the zeroth order AEM results were compared to be accurately the same (not depicted here for compactness). The previous realization serves to narrow the cause of code disparity down to the effect of the parameter γ in the model. Particularly, the thermodynamical plume expansion constant influences our model through the term $T_e(\tilde{n}^{\gamma-1}\epsilon^{j-1})$, multiplying the higher order density derivatives $\frac{\partial ln\tilde{n}^{i-j}}{\partial z}$ and $\frac{\partial ln\tilde{n}^{i-j}}{\partial r}$ in equations 3.10 and 3.11. This term is

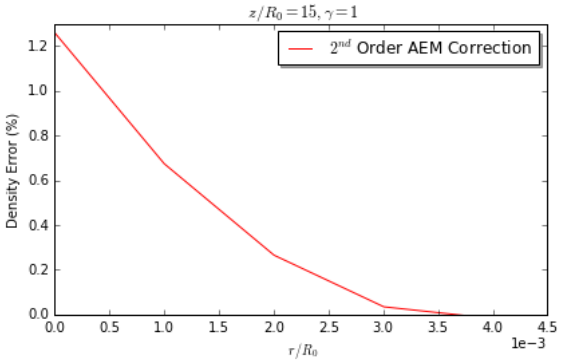
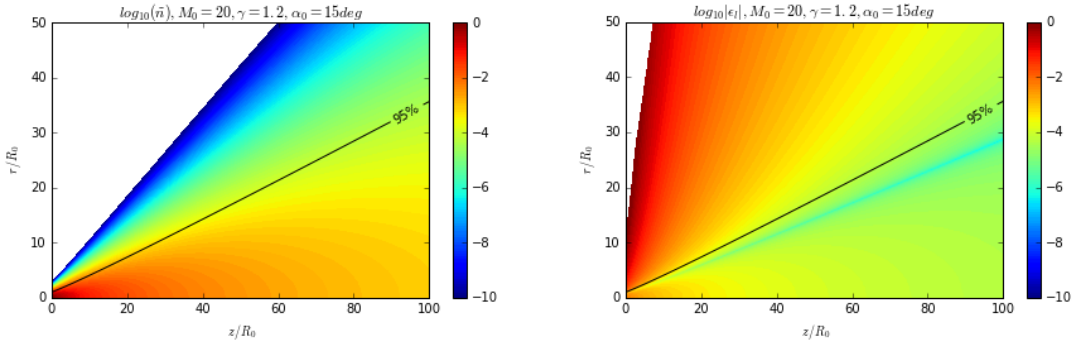


Figure 6.9: Density error comparison in regions close to the plume axis at the plume axial location $z = 15$

dropped in the cold beam solution approximation (zeroth AEM order), but affects the integration of higher AEM corrections. Thus, even if the mismatch shown in figures 6.7 and 6.8 is qualitatively small, there must exist numerical difference in the integration approaches between the codes.
Another possible cause of code dissimilarity is the calculation method of n, u_z, u_r variables along and near the plume axis, as commented in 5.4.1. Figure 6.9 shows the density error between the codes, in regions close to the center of the plume. Results obtained at further radial locations are obviously affected by the plume centerline outcome, by means of the $\frac{\partial ln\tilde{n}^{i-j}}{\partial r}$ term in equation 3.11. The realization of a possible integration difference affecting the results obtained from both codes is therefore, reinforced.

6.2 SSM Analysis

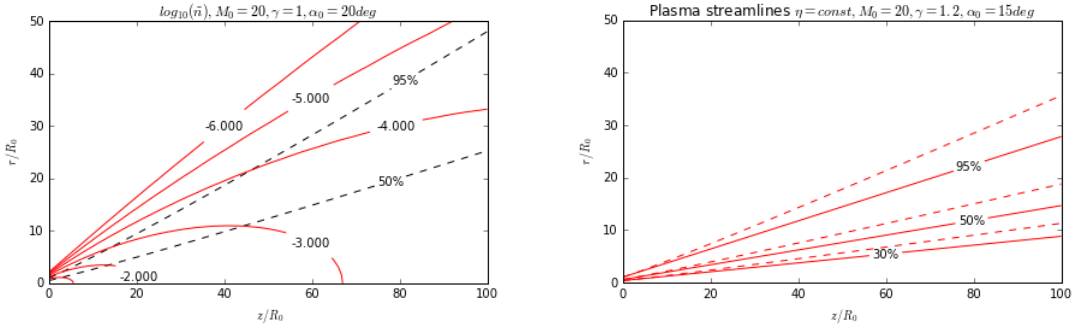
Generally, the SSM method has been used within a narrow range of plumes with particular initial density and velocity profiles. In previous papers, the Parks and Katz, Korsun and Tverdokhlebova or Ashkenazy and Frutchman particularizations were adopted to test the validity of the general SSM framework. Although the selected plume in this dissertation is one of such particularizations, to aid visualization and comparability, the new Python code used in the SSM model accepts any plasma profiles in compliance with equation 4.8 (More information on 5.4.2). In figure 6.10, the density and local SSM error plots of the selected plasma plume reproducing the Parks particularization,are presented. Notably, the abrupt change in the local error $log_1|\epsilon_l|$ pictured by the blue strip in the figure 6.10(b), which merely coincides with the ion streamline $\eta = \sqrt{(2/C)}$ where the local error committed by the SSM cancels out and sets the boundary for ϵ_l changing sign.



(a) (b)

Figure 6.10: Density logarithm contours (a) and the corresponding SSM model error ϵ_l (b), with the 95% ion current streamline $\eta = 1$ (black solid line) in both plots

It must be recalled that even though the SSM method is capable of simulating plumes with any initial density profile that complies with the physics of the model, the constraints imposed by equations 4.6 and 4.8 prevent the density and velocity to be independently defined by the user, and the AEM method should be used instead. Moreover the SSM works accurately under plasmas with initial conical velocity profiles, i.e $\delta^{'} = const$ only. These impositions explain the differences seen in 6.11b between the ion streamlines in the plasma, whether the SSM or the AEM is used.



(a) (b)

Figure 6.11: Plasma density logarithm contour plot (a) with $50\%, 95\%$ current ion streamlines (dashed black lines), and comparison on ion $30\%, 50\%, 95\%$ streamlines (b) between SSM (dashed lines) and the cold beam Zeroth Order AEM solution (solid red lines).

6.2.1 SSM validation against previous Plume Expansion Software

Analogously to 6.1.2, preceding SSM results were tested for correctness validation against previous Matlab software plume expansion results. Figure 6.12 visually summarizes the outcome of such validation tests.

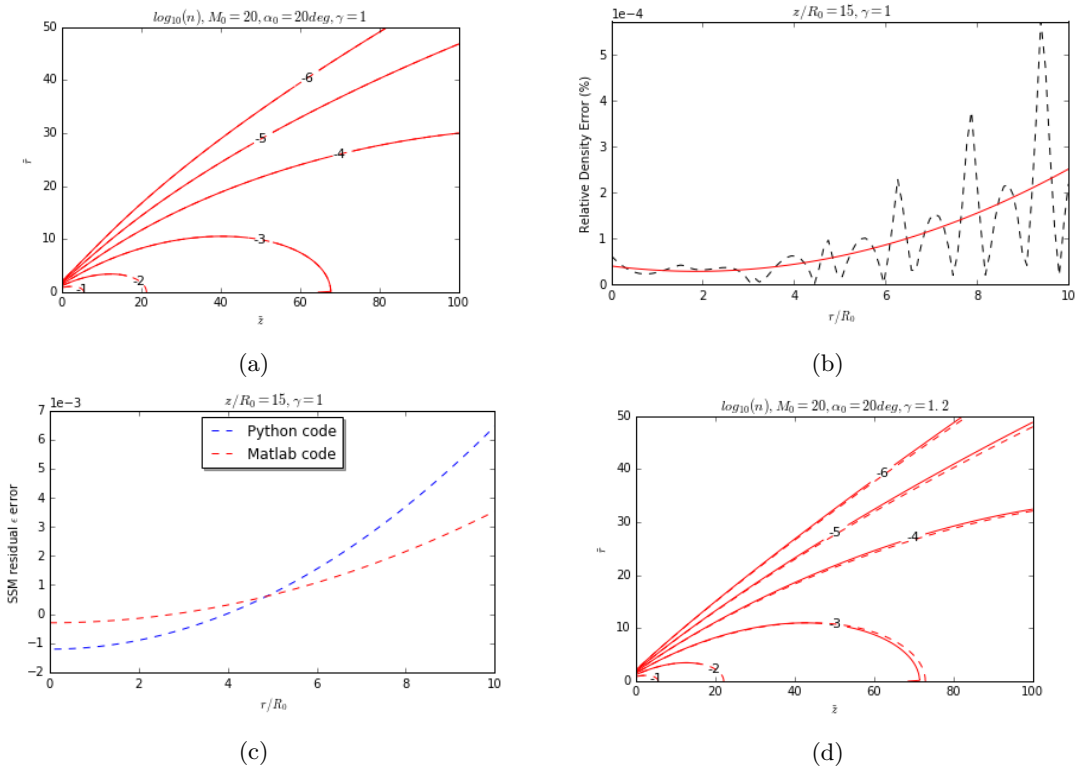
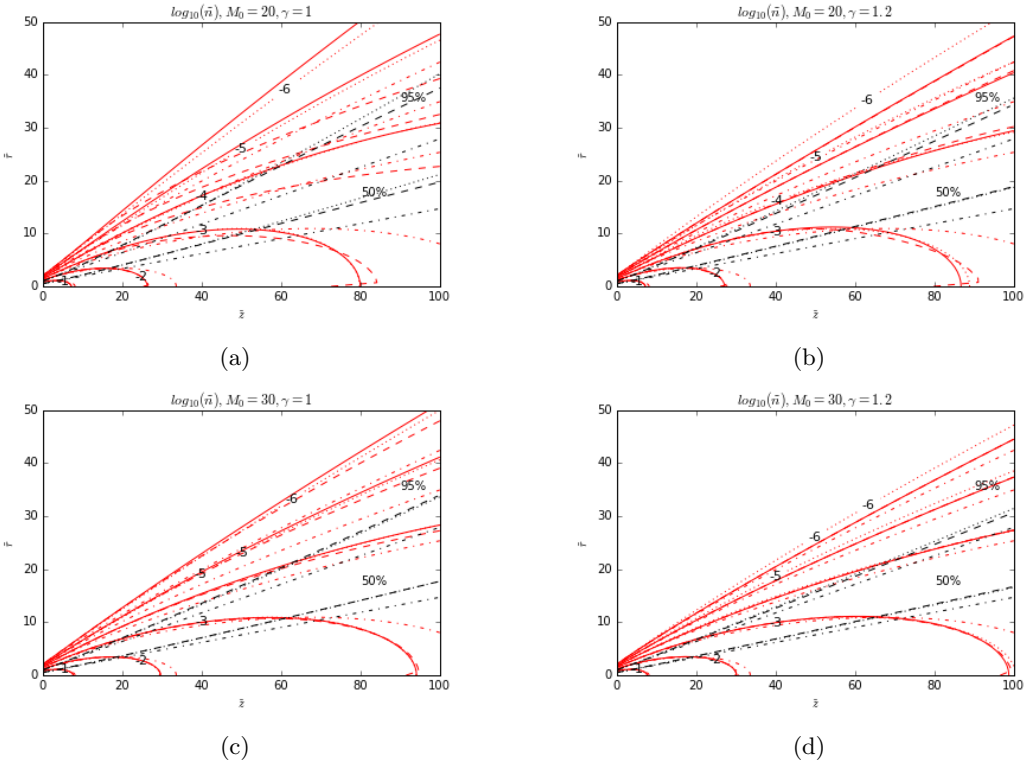


Figure 6.12: Comparison tests between Python plume Code and Matlab previous approach. Plot (a) depicts the plume density contours (logarithmic scale) obtained from Python (solid red lines) and Matlab (dashed red lines) codes, together with the 95% and 50% ion current streamlines (black lines). Plot (b) shows the relative error comparison on the returned plasma density values at $\tilde{z} = 15$ between the different plume codes, where the black dashed line represents raw data relative error and the smooth red line illustrates the best fit line of the results. Plot (c) compares the residual SSS error ϵ_1 for the same plume between the two codes.Finally, plot (d) compares plume axial velocity at $\tilde{z} = 60$ between Matlab and Python software.

Certainly, the returned outcome is fairly similar. Remarkably, the plasma density contours in plot (a), are nearly identical independently of which plume software is used. In fact, up to the calculation of $h_{\tilde{z}}$ from equation 4.12, the two codes are exactly equivalent. The integration therefore, diverges from this point in the model onwards, leading to differences in the remaining SSM plasma plume density or velocity variables, as observed in plots (b),(c) and (d).Although the differences are in the $10^{-3}, 10^{-4}$ order, it is important to detect the source of this divergence. Matlab Plume code does a somewhat exact imposition of density and velocity plasma profiles, following theoretical self-similar plasma models such as the ones reviewed in [19], [20] and [21]. Hence, equation 4.8 in the model is dropped for calculation purposes, and the SSM constraint derivations are not strictly followed in such approach. The effect is clearly seen in plot (d): while plume axial velocity $\tilde{u}_z$ is rigidly set along the whole radial domain in the plume front using the Matlab Code, values depart from this constant condition in the new Python Code. In any case, the variations are small enough to re-state the accuracy validness of the SSM Python code.

6.3 Comparison and discussion of the two models

A comparison on the accuracy between the SSM and AEM numerical solutions is presented, based on plume expansions with the previously reported far region initial profiles in figure 6.1. Both AEM second order corrections standard, and reinitialization results as well as SSM values are contrasted in the following pages, and as comparison basis the marching scheme AEM results are taken as common reference.
Figure 6.13 shows the contour map and the 50%-95% ion current lines of a plume with $\alpha_0 = 15$ and several combinations of M_0 and γ. As expected, the three methods described (AEM standard zeroth and second order corrections, AEM reinitialization second order correction, and SSM) behave similarly as plume Mach number increases especially in the case of non-isothermal plasmas. However, at lower M_0, the deviations in accuracy between the methods become visible, particularly further downstream in the plume: AEM baseline results seem to be a better fit at short distances, before the numerical error in the model grows excessively, while the SSM is better suited further downstream. This results were expected, since lower Mach numbers and plume expansions near to the isothermal limit, narrow down the convergence region in the AEM model and reduce the SSM accuracy as well. It is important to recall that both this methods rely on $M_0 \gg 1$ and therefore, the error depends on the plume Mach number (particularly on $1/M_0^2$) and disappears the cold beam limit $M_\infty \rightarrow$ is approached.



(a) (b) (c) (d)

Figure 6.13: Density (logarithmic) contours (red lines) and 50-95% ion streamlines (black lines) for a plume with $\alpha_0 = 15$ deg under different combinations of M_0 and γ. Different methods are shown, SSM(dotted lines), AEM zeroth order correction (dash-dotted lines), AEM second order correction (dashed lines) and AEM second order correction with reinitialization scheme (solid lines)

Figure 6.14 plots (a), (b), (c) and (d) show the relative errors in density with respect to the AEM marching scheme solution for the same plumes analyzed in figure 6.13. These plots were evaluated at $\tilde{z} = 100$ from the initial plane, to clearly visualize the radial evolution of the error at an axial distance where the methods do not behave optimally. Remarkably, it can be noted that at plume axis $\tilde{r} = 0$, the SSM density prediction is better regardless of the selected M_0 and γ. In addition, at low values of plume Mach number and a sufficient high radius, the error in the AEM second order correction becomes greater than the one achieved by the zeroth order solution. Indeed, this result is anticipated if one recalls that AEM higher order corrections fall out of the converging zone at ample radial distances from the plume centerline.

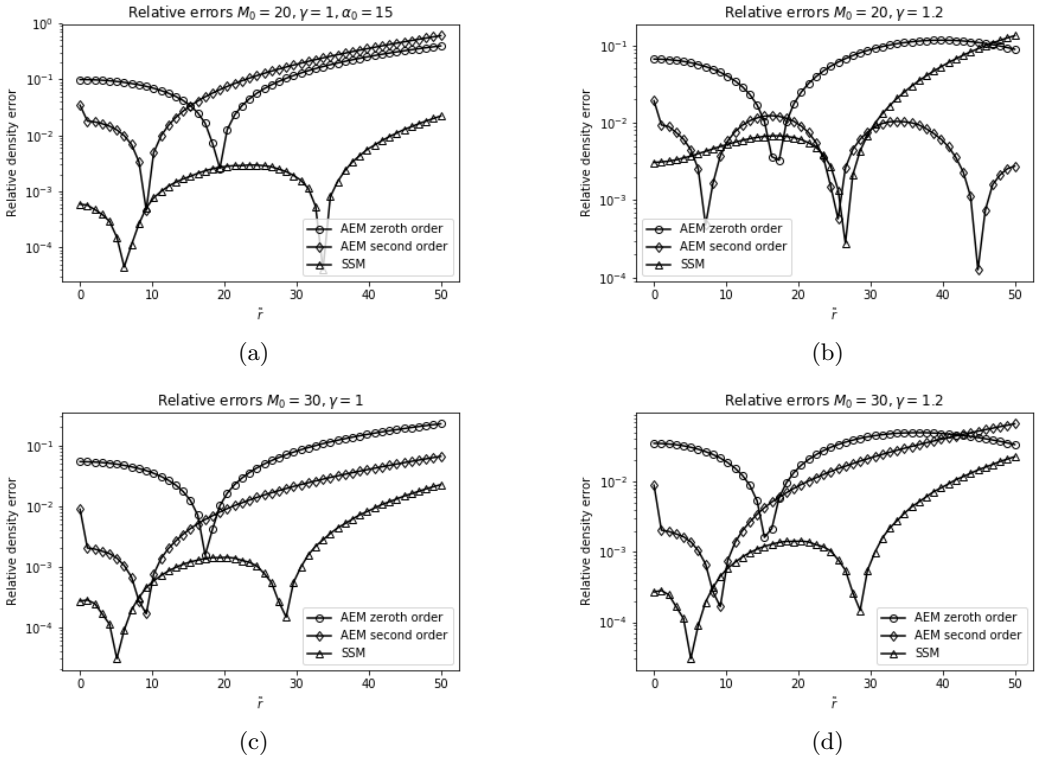


Figure 6.14: Relative errors with respect to the AEM marching scheme solution in density, for different combinations of M_0 and γ at distance $\tilde{z} = 100$ downstream of the plume.Three methods are shown and compared: Baseline AEM zeroth order (circles) and second order corrections (diamonds), as well as SSM results (triangles)

To conclude, both methods have their advantages and weaknesses. In principle, the AEM allows a wider selection of plasma plume expansions with exotic profiles such as the ones seen in HEMPT's (high-efficiency multistage plasma thrusters)[23] or DCFT (diverging cusped field thruster) [24] concepts. Finally, the SSM returns the ion current lines directly from the solution while AEM corrections terms are independent of M_0 and facilitate parametric studies of similar plume expansions.

Chapter 7

Discussion on plasma plume expansion

The results obtained from numerical simulations over these plume expansion models serve as stepping stone for supplementary studies of fundamental plasma magnitudes. In the following sections, an investigation on the physics of the plume ambipolar electric field ϕ and the far field divergence angle α_{FR}, as well as a brief discussion on the limitations of the Asymptotic and Self Similar models is presented.

7.1 Ambipolar electric field

The ambipolar electric potential $\tilde{\phi} = e\phi/T_{e0}$, induced by the electron expansion inside the plasma, is the only mechanism responsible for ion radial acceleration and plume divergence rise in the presented physical model. As electrons advance downstream with temperature T_e, an ambipolar electric field $-\nabla\phi \propto T_{e0}$ is created within the plume, confining such particles axially and radially. At the same time and simultaneously, the self-consistent response of the plasma to this physical effect results in the acceleration and radial branching (higher divergence angle) of ions downstream. Therefore, $\tilde{\phi}$ is the fundamental instrument in the energy transfer from electrons to ions. Figure 7.1 shows the evolution of $\tilde{\phi}$ in both SSM and AEM methods for different values of the thermal expansion constant γ, and η. It must be recalled, that the electric potential field depends on the full kinetic description of the electrons, which is in most cases a non trivial task. In this dissertation, a colisionless two-fuid plasma model is presented, and while the full fluid equations integrated from Vlasov's equation are valid, it is simpler to reduce the problem equations under some sort of parametric closure. Hence, the ambipolar electric field $-\nabla\phi$ is modeled under equation (2.10), i.e. under the assumption of an isotropic pressure tensor and an isothermal or polytropic expansion of the plasma. on these premises, the thermal expansion constant γ becomes and additional degree of freedom in the physical behavior of plumes, as observed from the dissimilar plume expansions with $\gamma = 1$ and $\gamma = 1.2$ in Figure 7.1. Nonetheless, this assumption is a mere approximation of the real colisionless plasma plume thermodynamical problem, and neglects the possibility of electron anisotropization leading to a loss of physical detail.
Although its extensive use in more complex expansion models, Boltzmann relation i.e., the isothermal expansion limit at $\gamma = 1$ (introduced in the presented model by means of equation (2.11)), yields the unphysical behavior of $\tilde{\phi} \to -\infty$ as $\tilde{n} \to 0$ downstream in the plume expansion. For hypersonic plumes where $M_0 \gg 0$ and consequently, the parameter $2\frac{\tilde{\phi}}{/M_0^2} = 2\frac{e\phi}{mu_{i0}^2}$ is made small, the $\gamma = 1$ limit leads to the unbounded acceleration of ions in the expansion as $u_{i0} \to \infty$. Moreover, the isothermal limit illustrating an infinite electron thermal conductivity cloud, restricts the computation of the plume-thruster energy balance (when $T_{e0} = const$ ions are accelerated boundlessly, the plasma requires an infinite thermal power supply from infinite electron heat fluxes). However, this unphysical behavior disappears when the plasma plume is allowed to cool down polytropically, with $\gamma > 1$. In these expansion cases, equation (2.11) sets the asymptotic limit of

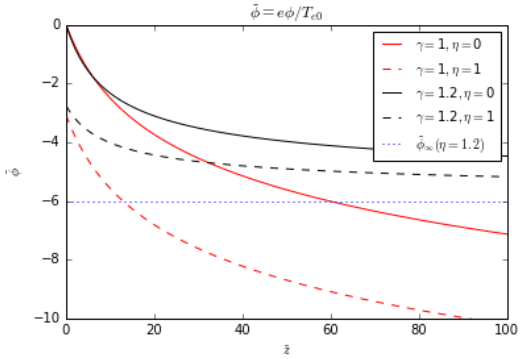


Figure 7.1: Evolution of ambipolar electric field $\tilde{phi}$ along the central plume axis $\eta = 0$ and the 95% ion current tube $\eta = 1$. The asymptotic value $\tilde{\phi}_\infty$ for polytropic expansions is shown in the light blue dotted line. Observe that isothermal and polytropic expansion models commence to diverge at approximately $\tilde{z} = 8$, along the central part of the plume

the plasma electric potential when $\tilde{n}, \tilde{T}_{e0} \to 0$ as:

$$\tilde{\phi}_\infty = -\frac{\gamma}{\gamma - 1} \tag{7.1}$$

This asymptotic limit can be observed as well in Figure 7.1.Again, the polytropic model seems to be a better match with the physics of plume expansions tested in laboratory.

7.2 Plume divergence angle

The analysis of the plume divergence angle is of central importance in the plume expansion process. As it was mentioned right at the beginning of this dissertation, high plume divergence angles do restrict the position of the thruster with respect to sensitive elements in the spacecraft, that can be damaged by plume particle impingement. In addition, a good estimation of the divergence angle can be used advantageously in fields such as space waste removal, since an accurate calculation can rapidly determine the momentum transfer towards an object downstream of the plume, as in the Ion Beam Shepherd technique.The only difficulty arises on how to correctly define a convention figure that allows the comparison and characterization of different plumes and thrusters. One might turn to the initial divergence angle α_0 to fully characterize the plum divergence, but this parameter would only be accurate for perfectly conical expansion plumes where $\alpha_0 = \alpha(\eta) = const.$ For any other non-conical plume the divergence angle changes in the far region, but keeps increasing downstream due to the combined effects of the residual thermal pressure and the ambipolar electric field. In these cases, as unifying parameter, the equivalent far plume field divergence angle is used, and defined

$$\alpha_{FR} = \frac{\tilde{R}_{95\%}(\tilde{z}_{FR}) - 1}{\tilde{z}_{FR}}, \tag{7.2}$$

as the half angle where the normalized radius $\tilde{R}_{95\%}(\tilde{z}_{FR})$ contains the 95% of the ion current streamline at a chosen $\tilde{z}_{FR}$ distance from the initial far region plane. Physically, this far region divergence angle is essentially influenced by the initial Plume Mach Number M_0,the initial divergence angle α_0, the effect of the electron pressure (modeled through the thermal expansion constant γ in this dissertation), and finally a small dependence on the initial density and velocity profiles.
Figure 7.2 shows the dependency of a_{FR} on these parameters, for both the AEM and SSM methods. From this figures several observations can be made: Of all the previously cited parameters,M_0 has strongest impact over the plume divergence. As the Mach number is increased, (by either achieving a faster plume through a greater voltage grid difference in the thruster,or by lowering the electron temperature with a well designed neutralizer), plume radial expansion declines due to the negligible

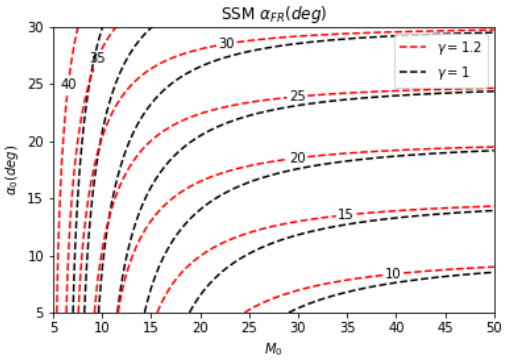


Figure 7.2: SSM Equivalent far region divergence angle α_{FR} at $\tilde{z} = 70$ map, as function of initial plume M_0 and α_0 and various values of γ

effect of electron thermal (pressure) effects. Ultimately, as $M_0 \to \infty$, α_{FR} approaches asymptotically towards the initial α_0.
Moreover, at any given value of α_0 and M_0 the far region divergence angle increases for lower values of γ, down to the isothermal limit $\gamma = 1$, due to the slower electron pressure decay.

Finally, the influence of the initial α_0 over the far region plume divergence angle is small,and even less pronounced when its value is already low.In view of such results, it is desirable to aim for high M_0 in the exhaust plume, rather than seeking for low initial plume divergence angles.
These results are obviously valid for any non-conical expansion plume. On the other hand, the value of α_{FR} would remain constant and equal to α_0 for the case of a cold conical plasma plume expansion.

7.3 Secondary and near region plume physics

Although semi-analytical two-fluid plume models (like the SSM or AEM presented extensively throughout this dissertation) are capable of tracking fundamental properties, some limitations are found in the physical study of plasma within such models. Study of the near region in plasma plumes, although being out of the scope of the dissertation, is important to determine a first estimate of far region initialization parameters, especially the initial plume divergence α_0.
Near-field plume momentum/charge exchange collisions affect the creation of slow stray ions that may diverge greatly from the plume axis line, therefore influencing the far region plume divergence α_0 and homogenizing n. In any case, these collisions become negligible further downstream in the plume and therefore, their effects can be omitted in far region plume simulations. Likewise, recombination collisions, can be disregarded over large distances due to their uncommon appearance, even when $\gamma > 1$ in fast rate cooling plumes.
Moreover, the presence of a dense background ambient plasma, neutrals and ambient magnetic fields distort the plume shape and may lead to wrong results and conjectures. As suggested in [25], oblique external magnetic fields $\mathbf{B}$ cause a compression of the plume cross section in the normal direction to both B and the plume axis, as well as the elongation of plume velocity in the plane defined by $\mathbf{B}$ and the plume axis. More precisely, external magnetics fields promote diamagnetic electric currents $\mathbf{j}$ inside the plasma, and in turn these currents induce an opposing plasma-generated magnetic field.The parameter $\beta_{total} = n(T_e + m_i u_i^2/2)(\beta^2/mu_0)$ derived in [25], compares the relative energy of this current induced field and the external one. Finally, the plasma currents are subjected to the additional Lorentz force $j \times \mathbf{B}$ that further deforms and affect plume divergence.
All the previously cited phenomena, important for spacecraft surface sputtering or contamination and ion flux determination, cannot be tackled with simple semi-analytical two fluid models. Note that a finer description of the lateral plume is necessary to that end, but radial error increases from plume axis in the AEM and SSM models. In these cases, Particle in Cell (PIC) codes are needed

to fully comprehend the full axial and lateral physics behind the plume expansion. As an example, the NOMADS-EP2 Plus 2D/3D hybrid-PIC code is being developed [4] to explore this additional plume phenomena.

Chapter 8

Conclusions

The physical behavior of hypersonic, two-fluid, quasi-neutral and collisionless plume expansion from a plasma thruster into vacuum has been studied under two approximate semi-analytical methods, the Asymptotic Expansion Method and the Self Similar Method. These two numerical solutions have been developed in Python code, a powerful open source language in both the academic and industrial fields. In addition, the overall structure of the code has been developed under a Class Oriented Object Programming approach and *PEP8* good programming practices standards. In this way, code modularization and reuse is being targeted to offer the user a quick and high-level usage of the code. Seeking freely distributable software and code sharing, the approximated solutions developed in this dissertation have been uploaded to *GitHub*, a web repository or version control system which manages and stores the revisions of projects interactively. Although free plume code is advantageous or all the scholar community, some precautions must be made to ensure open distribution among users yet acknowledging code authorship and licensing. To that end, a Digital Object Identifier has been attached to the the GitHub plasma plume code package to to make it easily and uniquely citeable, as well as the *MIT* permissive license which just requires preservation of copyright.

Back on the track of plume analysis, it has been made clear through this dissertation that either method, AEM or SSM, yield only approximation results of real plasma plume expansions and each one has advantages and weaknesses.

In terms of code usability and plume characterization, the AEM returns sensible results under a wider range of plumes with different initial density and velocity profiles therefore, being more flexible from the user perspective. Furthermore, AEM higher order corrections are independent of the expansion parameter ϵ promoting faster parametric studies on the effect of M_0 over the plume expansion, without the need to calculate this particular solutions each time. On the other hand, the constraints imposed in the SSM method (Eqs. (4.6) and (4.8)) limits the analysis to a narrower collection of simpler plumes where the solution needs to be re-worked at every Mach number. However, the SSM is mathematically straightforward and returns the ion current streamlines, by means of the dilation function h without further processing.

Regarding the accuracy of the solutions, both methods are specially well conditioned for hypersonic plumes with $M_0 \gg 1$. However, the most notable weaknesses of the AEM method are its narrow convergence zone and the need to include higher order corrections to increase the validity of the results, which in turn increments the complexity of the model equations. Moreover, the accuracy of the method seems to decline with γ, being critical in the case of isothermal plumes with $\gamma = 1$. Finally, these deficiencies can be remedied by the introduction of the Marching AEM Scheme at each subsequent integration or front step, at the cost of a more complex algorithm yet again. In contrast, the SSM contains the inherent error $\epsilon_l \propto 1/M_0^2$, which cannot be solved without leaving the condition of plume self-similarity itself.

The main parameters which control the plume expansion are the Mach number M_0,the electron thermodynamics modeled through the effective cooling rate γ, and the selection of initial far region profiles and especially α_0 . It has been shown that both the plasma ambipolar electric potential $\tilde{\phi}$ and the far field divergence angle α_{FR} are determined by these parameters. In the case of the ambipolar plasma potential, it has been deduced that the Boltzmann relation drawn from the

isothermal expansion limit $\gamma = 1$ proved to be physically deficient for infinite collisionless plume expansions, forcing the presence of some sort of cooling mechanism in these cases.
Additionally, the equivalent far region divergence angle α_{FR} seems to depend greatly on the choosing of M_0 and α_0. The Mach number opposes the effect of electron pressure in radial expansion and therefore, a higher plume M_0 leads to a decrease of plume far field divergence. On the other hand, α_0 sets a lower limit for the far field divergence. Finally, higher effective cooling rates, i.e $\gamma > 1$ lower the effect of radial expansion, thus affecting α_{FR} which has an upper value boundary when $\gamma = 1$. With this aspects in mind, an important performance gain during spacecraft and plasma thruster design can be achieved through the appropriate selection of plume exit Mach number and initial plume divergence. Especially in cases where Mach number and initial divergence is limited to low ranges, it can be more advantageous to increase M_0 rather than a decrease in α_0. To conclude, some additional features that were neglected in this current issue of the far region plume expansion model, but are nonetheless important from a physical perspective, were examined. Near plume field particle collisions, while unimportant in further region domains, shape the initial plasma and plume divergence profiles. Moreover, the presence of background ambient plasmas and external magnetic fields has been proved to distort the expansion of the plume and even its propagation direction under some special circumstances.

Socioeconomic Impact of Open-Source Plume Codes

Plasma plume expansion codes are already well in use within the academic community, as well as in private satellite integration and space exploration agencies. Although existing plume software may contain state-of-the-art methods based on deep plasma concepts and models, their content and accessibility is restricted to the more general public. In addition, these existing plume codes are far more complicated in computational resources as well as ease of use, than the code developed in HyperPlume. This dissertation does not attempt to alter the analysis of plume expansions by posing a new revolutionary software or advance plasma physical models, but contributes the most towards academic and the industrial satellite integration fields, by offering free and distributable plasma plume knowledge in the form of an Open-Source software (OOS).

Open-Source environments are thriving in business and company infrastructures, reducing code research and development costs.Figure 8.1 shows the total growth in open-source projects over recent years [29].The increase seems to be exponential. The benefits of introducing such open-source environments in the industry are not obscure at all either. Technology companies need to adapt fast to changes and trends, and developing new software from scratch is always suboptimal in the company working flow process. OOS´s allow companies to cut short software innovation and development times, by providing the grounds for further software customization. The original open-source project is in turn benefited from this customization, building a stronger standardized environment that can be reused by other technological commercial ventures. Businesses are

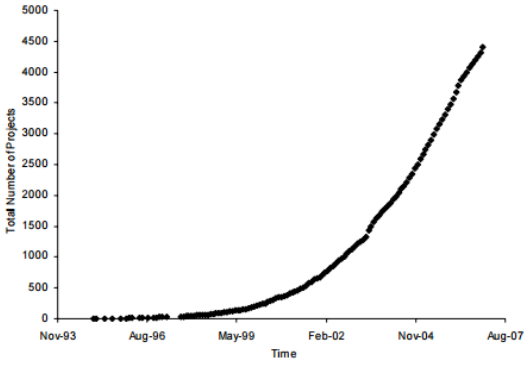


Figure 8.1: Total growth of Open-Source projects from Nov. 1993 to Aug 2007. [29]

therefore accelerated by the constant innovation and evolution of open source networks.
New entrepreneurs and companies in the satellite industry are emerging constantly, in the race to privatize space. In such rapid and changing field, newcomers can be greatly benefited by the quick physical insight provided in a plasma plume expansion open-source code. Plasma plume research is actively being conducted within the In-Space Propulsion Technologies framework [30], in order to evaluate spacecraft interactions, reduce the negative impact of exhaust plumes over the spacecraft and gain knowledge ans ultimately, decrease costs. Big space agencies or entities can be then assisted with fast plume simulations and physical results obtained from the open-source code presented in this dissertation, and use it as first version in the development of more complex tailored solutions. Lastly,from an academic perspective, new researchers gain fast and simple access to the field while experts feed deeper knowledge back to the project, increasing its value.

Future Work

The code developed in this dissertation integrates numerically two semi-analytical methods, AEM and SSM, for the plasma plume model derived in chapter 2. These methods,though returning fast accurate estimations of the physical effects behind a plume expansion, are approximations of the real problem at best. Hence, the code would be benefited from the introduction of a reference numerical integration method, that allowed relative comparison between the two semi-analytical methods. Purposely, the (exact) Method of Characteristics can be used to integrate numerically the model equations (2.13) to (2.15). The Method of Characteristics has been successfully implemented in [6] and [26] to study a wide range of plasma plume expansions, including magnetic effects. A MoC Python-open source code backed against the previous model is currently under development to provide benchmark results, and will be implemented in future revisions of the plasma plume package developed for this dissertation.
Regarding further upgrades and revisions to the AEM and SSM numerical codes, much can be still attained. As remarked in section 6.1.1, AEM results can be enhanced by introducing higher order corrections, following (3.10) to (3.13) equations. This in turn complicates the code from a programming perspective, so a detailed analysis on the pros and cons of such higher corrections must be done in future code corrections.
On the other hand, the proposed SSM code has been programmed over the valid, but still approximated condition $\tilde{u}_a = 1$ which simplifies model equations (4.7)-(4.9). SSM numerical code can be widened in future revisions by the alternative introduction of the ion energy equation at $\gamma = 0$,

$$\frac{1}{2}(\tilde{u}_c^2)' = -\frac{(\tilde{n}_c)^{\gamma-2}}{M_0^2}\tilde{n}_c',$$

which is advantageous in the analysis of the core region of plasma plumes as the SSM local error ϵ_l cancels out along the axis. Yet again, the introduction of this new assumption increases the complexity of the numerical code, since SSM model equations become coupled in $\tilde{u}_a, h$ and $\tilde{n}_a$.
Finally, on the topic of Python code scripting and software structure, the Object-Oriented approach can be enhanced to make the most out of test-driven programming and code reuse. In addition, coming revisions of the code could encapsulate the entire plume software package as a Python library, to bolster direct integration of the code in the user framework.

Dissertation Budget

The simulations and visualization results presented in this dissertation have been completed with the ASUS Notebook PC model **5511l**, containing a Intel(R) Core(TM) i7-4500U processor and 8 GB of RAM. The completion of this dissertation, code and simulations has expanded along a whole (academic) year due to limited simulation hours and the necessity to share academic time with professional time. In any case, code development took the central slotted time for dissertation completion, while simulations and result visualizations were fast forwarded.

Appendix A

Code Documentation

The complete description of the classes and methods included in the new plasma plume code are presented in this appendix chapter. This code documentation serves as a summarized user manual edited by one of the interactive and open source Python documentation tools, Sphinx. Sphinx allows code developers to automatically document Python code in manifold formats such as *HTML* and *PDF*. The following code guide was the output of Sphinx auto-documentation tool for Python Code, in *PDF* format and added to the dissertation for illustrative purposes.

Code Documentation

June 20, 2017

Table of Contents

1 Hyperplume class

class Hyperplume():

""" Parent class Hyperplume loads target plasma and defines common attributes as well as shared methods in the AEM and SSM plume classes"""

def solver(self):

"""Solver Abstract Method to be particularised by each Plume code. It is only defined for structure purposes in parent class Hyperplume"""

def query(self,z,r):

"""Query abstract method returns plasma profile data at specified grid points. query method is to be particularised by each plume code.It is only defined forstructure purposes in parent class Hyperplume"""

def __init__(self,plasma,z_span,r_span,n_init):

""" plume_constructor loads common class properties for AEM and SSM plume classes """

Args:

plasma (dict) simple_plasma object dictionary containing basic plasma parameters.

z_span (numpy.ndarray)

axial region where the problem will be integrated.

r_span (numpy.ndarray)

initial far-field plasma radial profile.

n_init (numpy.ndarray)

initial dimensional density front.

Usage:

```
>>> Plasma = {'Electrons': {'Gamma': 1,'T_0_electron':
2.1801714e-19,'q_electron': -1.6e-19},'Ions': {'mass_ion':
2.1801714e-25, 'q_ion': 1.6e-19}}
>>> z_span = np.linspace(0,100,100)
>>> r0 = np.linspace(0,3,100)
>>> n0 = np.exp(-6.15/2*r_span**2)
>>> Plume = Hyperplume(Plasma,z_span,r0,n0)
```

def simple_plasma(self,charge,ion_mass,init_plasma_temp,Gamma):

""" Method simple_plasma allows the user to quickly create a Plasma dictionary with two particle species (ions and electrons), and well defined attributes."""

Args:

charge (float) Electron charge given dimensional in units [C]

ion_mass (float) Ion mass given in dimensional units [Kg]

init_plasma_temp (float)

Initial plasma temperature given in dimensional units [J]

Gamma (int or float)

Dimensionless thermal expansion constant. Must be inside isothermal and polytropic boundaries [1,5/3]

Returns:

plasma (dict) Dictionary containing two simple plasma species (ions and electrons) with the before mentioned properties stored in favorable form.

Usage:

```
>>> Plasma = Hyperplume().simple_plasma(charge=1.6e-19,
ion_mass=2.1801714e-25,,init_plasma_temp=2.1801714e-19,Gamma=1)
```

def temp(self,n,n_0,T_0,Gamma):

""" Method temp calculates plasma temperature as function of plasma density."""

Args:

n (int or np.ndarray)

plasma density at specific (z,r) location in the plume grid
:n_0 (int):Iinitial density of plasma :T_0 (float): Initial temperature of plasma :Gamma (int): Dimensionless thermal expansion constant

Returns:

T (float or np.ndarray)

Temperature of plasma at targeted (z,r) grid points in plume

Usage:

```
>>> T =
Hyperplume().temp(n=0.65,n_0=1,T_0=2.1801714e-19,Gamma=1)
```

def phi (self,n,n_0,T_0,Gamma,e_charge):

"""Method phi calculates electric potential as function of plasma density."""

Args:

n(int or np.ndarray)

plasma density at specific (z,r) location in the plume grid
:n_0 (int):Iinitial density of plasma :T_0 (float): Initial temperature of plasma :Gamma (int): Dimensionless thermal expansion constant :e_charge (float):-Electron charge

Returns:

phi(float or np.ndarray)

Electric potential of plasma at (z,r) targeted grid point

Usage:

```
>>> phi = Hyperplume().phi(n=0.65,n_0=1,T_0=2.1801714e-19,
    Gamma=1,e_charge=-1.6e-19)
```

def n(self,n_0,T_0,phi,Gamma,e_charge):

"""Method n calculates plasma density as function of plasma potential """

Args:

:n_0 (int):Iinitial density of plasma :T_0 (float): Initial temperature of plasma :Gamma (int): Dimensionless thermal expansion constant
:e_charge (float):Electron charge

Returns:

n (float or numpy.ndarray)

Pasma density at (z,r) targeted grid point in the plume.

Usage:

```
>>> n = Hyperplume.n(n_0=1,T_0=2.1801714e-19,phi=-5.7
    Gamma=1,e_charge=-1.6e-19)
```

def eta_deriver(self,x,y):

"""Method eta_derivar calculates the numerical derivatives of the variables along eta, with a central finite difference approach"""

Args:

x (np.ndarray) represents the derivative step (dx,dy)

y (np.ndarray) vector to derive with respect to x

Returns:

y_prime (np.ndarray)

derivative of y over x stored in array format

Usage:

```
>>> x = np.array([0,0.5,1,1.2,2,2.3,2.6])
>>> y = np.array([10,17,23,27,36,40,45])
>>> dydx = Hyperplume.eta_deriver(x,y)
```

def plot(self,z,r,var_name,contour_levels):

""" Hyperplume Class method to plot the contours of important plasma variables along the specified (z,r) plume grid points"""

Args:

z (int,float, or np.ndarray)

new interpolation axial region where plasma variabes are to be calculated and plotted. Must be inside z_grid limits

r (int,float, or np.ndarray)

new interpolation axial region where plasma variabes are to be calculated and plotted. Must be inside z_grid limits

var_name (str) string containing the name of the variable to be visualized. Options are: 'lnn': logarithm of plasma density 'u_z': axial plume velocity 'u_r':radial plume velocity 'T': plasmaTemperature 'phi': ambipolar electric field 'eta': ion stream lines

contour_levels (array or of list): contour lables of plasma varialbled at the targets z,r points.

Usage:

```
>>> Plasma = Hyperplume().SIMPLE_plasma()
>>> Plume = AEM()
>>>
Plume.plot(z=np.array([15,20,25,30]),r=np.array([20,25,30,35]),
    var_name='n',contour_levels=[0,1,2,3,4,5,6,7,8])
```

2 SSM Class

class SSM(Hyperplume):

"""Self Similar model of a plasma plume expansion.Class SSM inherits methods __init__,-solver and query from parent class Hyperplume, and particularizes them."""

def __init__(self,plasma,M_0,d_0,z_span,r_span,n_init):

"""Constructor __init__ loads and initialises the main class attributes. Calls parent class Hyperplume constructor method __init__ to store main plasma properties as attributes in the class."""

Args:

plasma (dict) simple_plasma object dictionary containing basic plasma parameters.

z_span (numpy.ndarray)

axial region where the problem will be integrated.

r_span (numpy.ndarray)

initial far-field plasma radial profile.

n_init (numpy.ndarray)

initial dimensional density front.

M_0 (float) Plasma Mach number at (z,r) = (0,0)

d_0 (float) Tangent of initial plume divergence angle

Implementation:

```
>>> e_charge,ion_mass =1.6e-19,2.1801714e-25
>>> Plasma_temp,gamma_value = 2.1801714e-19,1
>>> Plasma = Hyperplume().simple_plasma(e_charge,ion_mass,
    Plasma_temp,gamma_value)
>>> z_span = np.linspace(0,110,5000)
>>> r_0 = np.linspace(0,10,5000)
>>> n0 =  np.exp(-6.15/2 * r_0**2)
>>> M0,d0=20,0.2
>>> Plume = SSM(Plasma,M0,d0,z_span,r_0,n0)
```

def solver(self):

"""Solver method solves for model constriants C and h, as well as the initial dimensionless axial velocity vector upsilon and initial dimensionless density profile nu, using SSM model equations. It then saves this plume variables as as class attributes, in the form of interpolation libraries over the entire plume grid.

Solver method is a particularization of the abstrac Hyperplume.solver() method """

Implementation:

```
>>> Plume.solver()
```

"""Important variables in solver method"""

C (float) SSM model separation constraint

h_interp (function)

SSM dilation function interpolation library

dh_interp (function)

Derivative of SSM dilation function h

nu_interp (function)

Dimensionless plasma density function

nu_prime_interp (function)

Derivative of Dimensionless plasma density function

upsilon_interp (function)

Dimensionless plume axial velocity interpolation library

"""To access the interpolation libraries and SSM constraints particularly:"""

```
>>> print(Plume.C,Plume.h)
>>> eta_target = 0.8
>>> Plume.nu_interp(eta_target)
>>> Plume.upsilon_interp(eta_target)
```

def dh_fun(h,Z):

"""dh_fun function calculates the derivative of the self-similar dilation function h(z), and saves the results as a class attribute in column-array format """

Args:

h (numpy.ndarray)

SSM model scaling function

Z (numpy.ndarray)

axial span for integration. Coincidet with initial axial span loaded in SSS class constructor,for accruacy and correctness)

Returns:

dh (numpy.ndarray)

derivative of SSM scaling function

def query(self,z,r):

""" Method query returns the density, velocity profile, temperature, the electric potential and SSM error at particular (z,r) points by interpolation over the Plume grid. SSM method query is a particulatization of the abstract Hyperplume method Hyperplume.query()"""

Args:

z (int,numpy.ndarray)

axial target points where plasma variables are retrieved. Single points, arrays of locations and meshgrids are valid.

r (int,numpy.ndarray)

axial target points where plasma variables are retrieved. Single points, arrays of locations and meshgrids are valid.

Returns:

lnn (int,numpy.ndarray)

logarithmic plasma density at specified (z,r) points in plume grid

u_z (int,numpy.ndarray)

plasma axial velocity at specified (z,r) points in plume grid

u_r (int,numpy.ndarray)

plasma radial velocity at specified (z,r) points in plume grid

T (int,numpy.ndarray)

plasma temperature at specified (z,r) points in plume grid

phi (int,numpy.ndarray)

plasma ambipolar electric potential at specified (z,r) points in plume grid

error (int,numpy.ndarray)

SSM error created by imposing model constraints at specified (z,r) points in plume grid

eta (int,numpy.ndarray)

ion current stream lines at specified (z,r) points in plume grid

Usage:

```
>>> z,r = np.linspace(0,100,50),np.linspace(0,50,40)
>>> lnn,u_z,u_r,T,phi,error,eta=Plume.query(z,r)
```

def type_parks(plasma,M_0,d_0,z_span,r_0,C):

""" type_parks functions allow the user to generate default plume density profiles based on the theoretical Parks plume model. The function creates the initial density profile following the theoretical model, and creates a SSM Plume object with unique characteristics"""

Args:

plasma (dict) Hyperplume's simple_plasma object, or otherwise a similar plasma dictionary containing basic parameters.

z_span (numpy.ndarray)

axial region where the problem will be integrated.

r_0 (numpy.ndarray)

initial far-field plasma radial profile.

M_0 (float) Plasma Mach number at (z,r) = (0,0)

d_0 (float) Tangent of initial plume divergence angle

C (float) SSM model constraint. C is a separation constant used for scaling the Self-Similarity plume proble. C is used to determine the initial density profile derived by Parks. In particular:

- n_parks = np.exp(-C*r_0**2 /2)

Returns:

Plume (object) SSM Plume object preloaded and solved with Parks theoretical density and axial velocity models.

Usage:

```
>>> Plasma = Hyperplume().simple_plasma(e_charge,ion_mass,
    Plasma_temp,gamma_value)
>>> z_span = np.linspace(0,110,5000)
>>> r_0 = np.linspace(0,10,5000)
>>> C = 6.15
>>>Plume_parks = type_parks(Plasma,z_span,r_0,C)
>>> lnn,u_z_,u_r,T,phi,error,eta=Plume_parks.query(z,r)
```

def type_korsun(plasma,M_0,d_0,Z_span,r_0,C):

""" type_parks functions allow the user to generate default plume density profiles based on the theoretical Korsun plume model. The function creates the initial density profile following the theoretical model, and creates a SSM Plume object with unique characteristics"""

Args:

plasma (dict) Hyperplume's simple_plasma object, or otherwise a similar plasma dictionary containing basic parameters.

z_span (numpy.ndarray)

axial region where the problem will be integrated.

r_0 (numpy.ndarray)

initial far-field plasma radial profile.

M_0 (float) Plasma Mach number at (z,r) = (0,0)

d_0 (float) Tangent of initial plume divergence angle

C (float) SSM model constraint. C is a separation constant used for scaling the Self-Similarity plume problem. C is used to determine the initial density and axial velocity profiles derived by Korsun. In particular:

- n_parks = 1 / (1 + C / 2 * r_0**2)
- upsilon_parks = (1 + C / 2 * r_0**2)**(gamma/2)

Returns:

Plume (object) SSM Plume object preloaded and solved with Parks theoretical density and axial velocity models.

Usage:

```
>>> Plasma = Hyperplume().simple_plasma(e_charge,ion_mass,
    Plasma_temp,gamma_value)
>>> z_span = np.linspace(0,110,5000) # Axial plume grid for
integration
>>> r_0 = np.linspace(0,10,5000) #Initial plume radial profile
>>> C = 6.15
>>>Plume_korsun = type_korsun(Plasma,z_span,r_0,C)
>>> lnn,u_z_,u_r,T,phi,error,eta=Plume_korsun.query(z,r)
```

def type_ashkenazy(plasma,M_0,d_0,Z_span,r_0,C):

""" type_ashkenazy functions allow the user to generate default plume density profiles based on the theoretical ashkenazy plume model. The function creates the initial density profile following the theoretical model, and creates a SSM Plume object with unique characteristics"""

Args:

plasma (dict) Hyperplume's simple_plasma object, or otherwise a similar plasma dictionary containing basic parameters.

z_span (numpy.ndarray)

axial region where the problem will be integrated.

r_0 (numpy.ndarray)

initial far-field plasma radial profile.

M_0 (float) Plasma Mach number at (z,r) = (0,0)

d_0 (float) Tangent of initial plume divergence angle

C (float) SSM model constraint. C is a separation constant used for scaling the Self-Similarity plume problem. C is used to determine the initial density profile derived by Ashkenazy. In particular:

- n_parks = (1 + k*r_0**2)**(-C/(2*k))
- upsilon_parks = (1 + k*r_0**2)**(-1/2), where k = d_0**2

Returns:

Plume (object) SSM Plume object preloaded and solved with Parks theoretical density and axial velocity models.

Usage:

```
>>> Plasma = Hyperplume().simple_plasma(e_charge,ion_mass,
    Plasma_temp,gamma_value)
>>> z_span = np.linspace(0,110,5000)
>>> r_0 = np.linspace(0,10,5000)
>>> C = 6.15
>>>Plume_ashk = type_ashkenazy(Plasma,z_span,r_0,C)
>>> lnn,u_z_,u_r,T,phi,error,eta=Plume_ashk.query(z,r)
```

3 AEM Class

class AEM(Hyperplume):

"""Asymptotic Expansion Model of a plasma plume expansion.Class AEM inherits methods from parent class Hyperplume, and particularizes them.All initial inputs must be given in dimensional form."""

def __init__(self,plasma,z_span,r_span,n_init,uz_init,ur_init,sol_order):

""" Class method __init__ is used as class constructor. Calls parent class Hyperplume constructor method __init__ to store main plasma properties as attributes in the class."""

Args:

plasma (dict) simple_plasma object dictionary containing basic plasma parameters.

z_span (numpy.ndarray)

axial region where the problem will be integrated.

r_span (numpy.ndarray)

initial far-field plasma radial profile.

n_init (numpy.ndarray)

initial far-field plum density front.

uz_init (numpy.ndarray)

initial far-region plume axial velocity profile

ur_init (numpy.ndarray)

initial fr-region plume radial velocity profile

sol_order (int) Integer defining the AEM correction order for the plume integration. -0: AEM "cold beam" zeroth order solution -1: AEM first order correction -2: Second Order Correction

Usage:

```
>>> e_charge,ion_mass = 1.6e-19
>>> Plasma_temp,gamma_value= 2.1801714e-25,2.1801714e-19,1
>>> Plasma = Hyperplume().simple_plasma(e_charge,ion_mass,
    Plasma_temp,gamma_value)
>>> z_span = np.linspace(0,110,5000)
>>> r_0 = np.linspace(0,10,5000)
>>> n0 =  np.exp(-6.15/2 * r_0**2)
>>> uz0,ur0  =
np.linspace(20000,20000,100),np.linspace(0,40000,100)
>>> AEM_Order = 2 # AEM model solution
>>> PlumeAEM = AEM(Plasma,z_span,eta_0,n0,uz0,ur0,AEM_Order)
```

"""Other important class attributes loaded in the AEM constructor are:"""

d0 (numpy.ndarray)

far field initial divergence ur0/uz0.

d0p (numpy.ndarray)

derivative of plume initial divergence

eps (float) AEM expansion parameter 1/M_{0}^{2}

uz0p (numpy.ndarray)

derivative of initial far region axial velocity

duz0p (numpy.ndarray)

derivative of initial far region radial velocity

z_grid,r_grid (numpy.ndarray)

Plume grids where AEM problem is integrated

To access these attributes, for instance:

```
>>> print(PlumeAEM.d0)
>>> print(PlumeAEM.eps)
```

def solver(self):

""" Class method Solver integrates the AEM model equations in the specified plume grid. The method stores the different order plasma properties in matrixes of size (mxn), where m,n are the number of z,r points,respectively.Porperties such as density,temperature,electric field,etc are calculated and saved as attributes of the class in this matrix form."""

Usage:

```
>>> PlumeAEM = AEM(Plasma,z_span,eta_0,n0,uz0,ur0,AEM_Order)
#Creation of AEM plume
>>> PlumeAEM.solver() # be sure to create a valid AEM plasma plume
before applying the plume solver method
```

"""Main Plume properties solved and saved by the method as class attributes:"""

lnn (numpy.ndarray)

3-D matrix containing density values (logarithmic) for the three possible AEM solution orders.

uz (numpy.ndarray)

3-D matrix containing axial velocity values for the three possible AEM solution orders.

ur (numpy.ndarray)

3-D matrix containing radial velocity values for the three possible AEM solution orders.

T (numpy.ndarray)

3-D matrix containing plasma Temperature values for the three possible AEM solution orders.

:phi (numpy.ndarray):3-D matrix containing plasma ambipolar electric field for the three possible AEM solution orders. :div (numpy.ndarray): 3-D matrix containing plume divergence values for the three possible AEM solution orders. :eta_ (int,numpy.ndarray): 3-D matrix containing ion current streamlines for the three possible AEM solution orders.

```
>>> PlumeAEM.lnn[0,:,:] # density values for Zeroth Order AEM
>>> PlumeAEM.uz[1,:,:] # axial velocity First Order AEM
>>> PlumeAEM.T[2,:,:] # Temperature values for Second Order AEM
```

def marching_solver(self,nsects):

"""Marching schem AEM plume solver.AEM Class method marching_solver solves extends the AEM solution downstream by reinitializing the method at each new calculated plasma plume front, (r0_front,z0_front,uz0_front,n0_front) preventing excessive error growth in the calculations and widening the convergence region of thr AEM model.

Marching_solver method reinitializes the plume initial parameter, with the values calculated in the previous integration step, as many times as indicated by the user in nsects. It then solves the plume expansion incrementally by callling the solver method multiple times."""

Args:

nsects (int) number of axial sections or steps (plume fronts), where solver reinitializes the model and integrates the solution agin.

Usage:

```
>>> PlumeAEM = AEM(Plasma,z_span,eta_0,n0,uz0,ur0,AEM_Order)
#Creation of AEM plume
>>> Plume.marching_solver(nsects=100)
```

"""Same Plasma attributes from standard solver can be accessed in Method marching_solver, but in this case the method stores only the ith higher order correction specified by the user at plume creation with the input argument sol_order:"""

lnn (numpy.ndarray)

matrix containing density values (logarithmic) for the selected AEM solution order.

uz (numpy.ndarray)

matrix containing axial velocity values for the selected AEM solution order.

ur (numpy.ndarray)

matrix containing radial velocity values for the selected AEM solution order.

T (numpy.ndarray)

matrix containing plasma Temperature values for the selected AEM solution order.

phi (numpy.ndarray)

matrix containing plasma ambipolar electric field for the selected AEM solution order.

div (numpy.ndarray)

matrix containing plume divergence values for the selected AEM solution order.

```
>>> PlumeAEM.lnn # density values for AEM ith order solution of
plume expansion in the grid
>>> PlumeAEM.uz # axial velocity for AEM ith order solution
>>> PlumeAEM.T # Temperature values for AEM ith order solution
```

def query(self,z,r):

""" Method query returns the density, velocity profile, temperature, the electric potential at particular (z,r) points in the Plume.

These plasma properties are interpolated along the previously calculated 2D grids z_grid and r_grid at targeted (z,r) points specified by the user. User must always check if np.max(r) > np.max(self.r_grid), np.max(z) > np.max(self.z_grid) in their query point set,to avoid extrapolation results."""

Args:

z (float,numpy.ndarray)

new interpolation z points.

r (float,numpy.ndarray)

new interpolation r points.

Outputs:

lnn (int,numpy.ndarray)

logarithmic plasma density at specified (z,r) points in plume grid

u_z (int,numpy.ndarray)

plasma axial velocity at specified (z,r) points in plume grid

u_r (int,numpy.ndarray)

plasma radial velocity at specified (z,r) points in plume grid

T (int,numpy.ndarray)

plasma temperature at specified (z,r) points in plume grid

phi (int,numpy.ndarray)

plasma ambipolar electric potential at specified (z,r) points in plume grid

eta (int,numpy.ndarray)

ion current stream lines at specified (z,r) points in plume grid

Usage:

```
>>> z,r = np.linspace(0,100,50),np.linspace(0,50,40) #target (z,r)
for plume query
>>> lnn,u_z,u_r,T,phi,eta=PlumeAEM.query(z,r)
```

def grid_setup(self,zpts,epts):

""" grid_setup creates an strctured grid of z,r points where the AEM problem will be integrated """

Args:

zpts (int) number of axial points in the structure. Indicates legnth oof axial plume span

epts (int) number of radial points in the structure. Indicates legnth of radial plume span

Returns:

z_grid (numpy.ndarray)

2D matrix containing axial grid points for model integration

r_grid (numpy.ndarray)

2D matrix containing radial grid points for model integration

Usage:

```
>>> z_grid,r_grid = PlumeAEM.grid_setup(100,50)
```

def val_domain(self):

"""val_domain class method evaluates the validity of the AEM series expansion solution at z_grid and r_grid points in the plume. Validity results for each AEM order are stored in the 3D matrix Plume.val. These matrix is filled with values indicating a specific validity condition in the results.

Validity Values

- 0 - Not valid for both velocity and density

- 1 - Valid only for velocity
- -1 - Valid only for density
- 2 - Valid for both velocity and density

"""

Usage:

```
>>> PlumeAEM.val_domain() #Intialize validity condition study
>>> print(Plume.val) #See results of validation
```

def partial_derivs(self,var,type2):

"""Class method partial_derivs computes the partial derivatives of plasma variables with respect to the physical z,r at the plume grid points."""

Args:

var (numpy.ndarray)

Variable values to derive at z,r grid points

type2 (int) Integer defining the behaviour of the derivative at the borders and therefore the Type of varible to be differentiated.

- 0: Symmetric function. Border derivative value is set to 0
- -1: Anti-symmetric function. Forward finite difference is used for border derivative calculation.

Returns:

dvar_dz (numpy.ndarray)

z-partial derivative values of input argument var at the grid points

dvar_dr (numpy.ndarray)

r-partial derivative values of input argument var at the grid points

Usage:

```
>>> dlnn0_dz,dlnn0dr = PlumeAEM.partial_derivs(Plume.lnn_0)
#derivative of Zeroth order density correction
```

Bibliography

[1] George P. Sutton,Oscar Blibarz, *Rocket Propulsion Elements,8th edition.* Wiley, John Wiley & Sons, Inc,2010.

[2] Merino M, Ahedo E, Bombardelli C, Urrutxua H and Peláez J, *Ion beam shepherd satellite for space debris removal*, Progress in Propulsion Physics (EUCASS Advances in Aerospace Sciences vol 4) ed L T DeLuca et al (Moscow:Torus) chapter 8, pp 789–802.

[3] M. Merino, E. Ahedo, C. Bombardelli, H. Urrutxua, and J. Peláez., *Hypersonic plasma plume expansion in space*, In 32nd International Electric Propulsion Conference, number IEPC-2011-086, Fairview Park, OH, 2011. Electric Rocket Propulsion Society.

[4] F. Cichocki,M.Merino,E.Ahedo,*Modeling and Simulation of EP Plasma Plume Expansion into Vacuum* Propulsion and Energy Forum, American Institute of Aeronautics and Astronautic, 50th AIAA/ASME/SAE/ASEE Joint Propulsion Conference, 2014.

[5] S. H. Lam, *MAE 558 Plasmadynamics Notes* Department of Mechanical and Aerospace Engineering Princeton University,March 12, 1997.

[6] F. Cichocki,M.Merino,E.Ahedo, *A collisionless plasma thruster plume expansion model* Plasma Sources Science and Technology, IOP Publishing, 2015.

[7] Myers R. and Manzella D. 1993, *Stationary plasma thruster plume characteristics* at the 23rd Int. Electric Propulsion Conf. (Fairview Park, OH: Electric Rocket Propulsion Society) IEPC 93–096

[8] Absalamov S. et. al. 1992, *Measurement of plasma parameters in the stationary plasma thruster (SPT-100) plume and its effect on spacecraft components.* 28th AIAA/ASME/SAE/ASEE Joint Propulsion Conf. (Washington, DC: AIAA) AIAA 92–3156

[9] Foster J., Soulas G. and Patterson M. *2000 Plume and discharge plasma measurements of an NSTAR-type ion thruster* at 36th AIAA/ASME/SAE/ASEE Joint Propulsion Conf. and Exhibit(Washington, DC: AIAA) AIAA 2000–3812

[10] Dannenmayer K., Mazouffre S., Ahedo E. and Merino M. 2012 *Hall effect thruster plasma plume characterization with probe measurements and self-similar fluid models*, at 48th AIAA/ASME/SAE/ASEE Joint Propulsion Conf. and Exhibit (Washington, DC: AIAA) AIAA 2012–4117.

[11] : Beal B., Gallimore A. and Hargus W. 2005, *Phys. Plasmas* 17 073501

[12] Goebel D. and Katz I. 2008 *Fundamentals of Electric Propulsion: Ion and Hall Thrusters* (New York: Wiley).

[13] Celik M., Santi M., Cheng S., Martínez -Sánchez M. and Peraire J. 2003, *Hybrid-PIC simulation of a Hall thruster plume on an unstructured grid with DSMC collisions* at 28th Int. Electric Propulsion Conf. (Fairview Park, OH: Electric Rocket Propulsion Society) IEPC 03–134

[14] Dannenmayer K., Mazouffre S., Ahedo E. and Merino M. 2012, *Hall effect thruster plasma plume characterization with probe measurements and self-similar fluid models* at 48th AIAA/ASME/SAE/ASEE Joint Propulsion Conf. and Exhibit (Washington, DC: AIAA) AIAA 2012–4117

[15] J. Navarro,and M. Merino, and E. Ahedo *Two-Fluid and PIC-Fluid Code Comparison of the Plasma Plume in a Magnetic Nozzle* at 48th AIAA/ASME/SAE/ASEE Joint Propulsion Conference.Exhibit 30 July - 01 August 2012, Atlanta, Georgia.

[16] : Myers R. and Manzella D. 1993, *Stationary plasma thruster plume characteristics* at 23rd Int. Electric Propulsion Conf. (Fairview Park, OH: Electric Rocket Propulsion Society) IEPC 93–096. [10] Beal B., Gallimore A. and Hargus W. 2005, *Phys. Plasmas* 17 073501

[17] Goebel D. and Katz I. 2008 *Fundamentals of Electric Propulsion: Ion and Hall Thrusters* (New York: Wiley).

[18] E.J. Hinch. *Perturbation methods.* Cambridge University Press, 1991

[19] DE Parks and I. Katz, *A preliminary model of ion beam neutralization.* In 14th International Electric Propulsion Conference, Fairview Park, OH, 1979. Electric Rocket Propulsion Society.

[20] 4A.G. Korsun and E.M. Tverdokhlebova., *The characteristics of the EP exhaust plume in space.* In 33rd AIAA/ASME/SAE/ASEE Joint Propulsion Conference Exhibit, Washington DC, 1997. AIAA.

[21] J. Ashkenazy and A. Fruchtman., *Plasma plume far field analysis.* In 27th International Electric Propulsion Conference, Fairview Park, OH, 2001. Electric Rocket Propulsion Society.

[22] Anatoly G. Korsun,Ekaterina M. Tverdokhlebova†,Flur F. Gabdullin,*The Distinction between the EP Plume Expansion in Space and in Vacuum Chamber*.Presented at the 29th International Electric Propulsion Conference, Princeton University,October 31 – November 4, 2005.

[23] KochK and SchirraM 2011, *The HEMPT concept: a survey on theoretical considerations and experimental evidences* at 32nd Int. Electric Propulsion Conf. (Fairview Park, OH: Electric Rocket Propulsion Society).IEPC 2011–236

[24] AG Korsun, EM Tverdokhlebova, and FF Gabdullin., *Mathematical model of hypersonic plasma flows expanding in vacuum.* Computer physics communications, 164(1-3):434–441, 2004.

[25] AG Korsun, EM Tverdokhlebova, and FF Gabdullin., *Mathematical model of hypersonic plasma flows expanding in vacuum.* Computer physics communications, 164(1-3):434–441, 2004.

[26] M. Merino, E. Ahedo, *Toberas Magnéticas para Motores Espaciales de Plasma.* Proyeco Fin the Carrera at Escuela Técnica Superior de Ingenieros Aeronaúticos, February 2010.

[27] Brenning N., Hurtig T. and Raadu M. 2005, *Phys. Plasmas* 12 012309

[28] BrenningN, HurtigT and RaaduM 2005, *Phys. Plasmas* 12 012309.

[29] Amit Deshpande,Dirk Riehle, *The Total Growth of Open Source.* Proceedings of the Fourth Conference on Open Source Systems (OSS 2008). Springer Verlag, 2008.

[30] *NASA Technology Roadmaps,TA 2: In-Space Propulsion Technologies.*National Aeronautics and Space Administration,July 2015.

[31] https://github.com/Pabsm94/HyperPlume