A Continuum Kinetic Investigation into the Role of Transport Physics in the Bohm Speed formulation.

Vignesh Krishna Kumar

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Aerospace Engineering

Bhuvana Srinivasan, Chair

Colin S. Adams

Wayne A. Scales

September 1, 2023

Blacksburg, Virginia

Keywords: Plasma Sheath, Continuum Kinetics, Plasma Transport , Bohm Speed, Computational Plasma Dynamics

A Continuum Kinetic Investigation into the Role of Transport Physics in the Bohm Speed formulation.

Vignesh Krishna Kumar

(ABSTRACT)

When plasmas come in contact with the boundaries that confine them, various complex processes occur between the plasma and the materials in the boundary. These processes, called plasma-material interactions (PMI) lead to physical and chemical modifications in the materials and in the plasma. In the case of a tokamak, a magnetic confinement fusion reactor, the interactions between the plasma and the material in the bounding walls can negatively impact the performance and service life of the reactor. Furthermore, PMI are also found in other areas of significant engineering interest, such as plasma-based spacecraft propulsion engines, where interactions affect the transport properties of the plasma and consequently the performance of the engine. Therefore, gaining a fundamental understanding of the various plasma-material interactions is necessary for the development and improvement of these devices.

PMI are dictated by the plasma sheath, a layer of net positive charge that forms at the plasma-boundary interface. The sheath regulates the energy and particle fluxes to the boundary, mediating the interactions. Sheaths, however, are only stable and well-developed when the ions enter the sheath with a speed equal to or greater than the 'Bohm speed'. The Bohm speed is a landmark result in sheath theory and various mathematical expressions for it have been derived from fluid and kinetic treatment of plasmas. Although these models are widely used, they are only accurate in cases where the thickness of the sheath is negligible when

compared to the scale length of the plasma in consideration. These cases are said to satisfy the 'asymptotic limit'.

To resolve this, a new Bohm speed model that considers the effects of transport terms such as the electron heat flux, thermal force, and temperature isotropization has been recently proposed [Y. Li et al., Physical Review Letters (2022)]. The model is verified using particle-in-cell (PIC) kinetic simulations and is shown to accurately predict the Bohm speed in cases away from the asymptotic limit. This thesis investigates the new model using the continuum kinetic approach on the Gkeyll software framework. The continuum kinetic approach numerically solves the Vlasov-Maxwell equations using the discontinuous Galerkin method and captures the kinetic phenomena of the plasma without needing to track individual particles. Multiple collisional cases ranging from a Knudsen number of 20 to 5000 are considered in a 1X3V simulation domain using the Lenard-Bernstein collisional operator.

The results of the continuum kinetic simulations are benchmarked to the PIC simulation results. It is concluded that across a wide range of collisionalities, the continuum kinetic method captures much of the same physics as the PIC method while offering noise-free results. However, there is a discrepancy between the Bohm speed prediction and the simulation results in the continuum kinetic case. This discrepancy is explored and significant error in the collisional integral derived transport terms between the continuum kinetic method and PIC method is found, suggesting that the difference in collisional operator may be the source of the discrepancy. Nevertheless, the sheath profiles developed in the PIC simulations and the continuum kinetic simulations are in reasonable agreement.

A Continuum Kinetic Investigation into the Role of Transport Physics in the Bohm Speed formulation.

Vignesh Krishna Kumar

(GENERAL AUDIENCE ABSTRACT)

Nuclear fusion is a process in which two light atomic nuclei (like hydrogen) fuse to form a heavier nucleus (like helium) and release tremendous amounts of energy. The resultant energy from these reactions powers the sun and has the potential to provide clean energy for our terrestrial needs. Harnessing fusion energy has been one of the biggest scientific and engineering challenges of our time due to various reasons. One of these reasons is the interaction of plasma, which is the fuel for the fusion reaction, and the materials of the walls that bound the plasma. These interactions are called plasma-material interactions (PMI) and can affect the longevity and performance of fusion reactors.

The main governing phenomenon behind these interactions is the plasma sheath, a layer of plasma that is formed when the plasma encounters a boundary. For a sheath to form it is also necessary that ions in the plasma possess a speed greater than the so-called 'Bohm speed'. While many expressions have been derived for the Bohm speed, these expressions are only valid when there is a clear sheath entrance that divides the bulk plasma and the sheath. This condition is not satisfied in many cases of interest. Instead, a sheath-transition region is found to exist between the bulk plasma and the sheath.

A recently proposed Bohm speed model [Y. Li et al., Physical Review Letters (2022)] resolves this. This model is accurate in cases where the sheath-transition region exists and

is derived by considering previously overlooked transport physics. In this work, this new model is studied using a different computational approach known as 'continuum kinetics' using an open-source solver called Gkeyll. The results of the continuum kinetic simulations are compared to the results used to verify the model.

Dedication

ബാലനും രാജിക്കും സമർപ്പിക്കുന്നു

Acknowledgments

While I am the sole author of this work, its completion was in no way a sole effort. This thesis was made possible through the invaluable assistance and support of many individuals who have stood alongside me throughout the course of my master's degree.

First of all, I would like to express my deepest gratitude to my advisor, Professor Bhuvana Srinivasan. Throughout my master's journey, Dr. Srinivasan has served as a source of continuous support, guidance, and encouragement. She believed in me, even in times I could not, and inspired me to persevere during difficult times. Her seemingly endless patience allowed me to enjoy my research and appreciate the world of plasma physics. I also extend my sincere thanks to the members of my thesis committee, Professor Wayne Scales and Professor Colin Adams, for their time, constructive feedback, and critical review of my work.

In addition, I would like to recognize all the wonderful members of our Plasma Dynamics Computational Laboratory here at Virginia Tech, past and present. It has been an incredible experience working with this group and I could not have asked for more approachable, friendly, and helpful colleagues. In particular, I would like to thank Kolter Bradshaw, for being a constant source of guidance since day one; Chirag Skolar, for being my mentor and for our weekly meetings; Yuzhi Li, for always being willing to help.

I would like to especially thank some of the faculty and staff that I have had the joyous opportunity to interact with at Virginia Tech, who have always been kind and eager to assist me. Namely, Dr. Aurelien Borgoltz, John Burleson, Adey Kefenie, Erin Wilson. I would

also like to acknowledge the Advanced Research Computing (ARC) at Virginia Tech for providing the computational resources and technical support that contributed to the results of this thesis.

On a more personal note, a heartfelt thank you to my dear friends at VT for keeping me sane for the past two years. Sriya, thank you for being on my team, for always listening to me, for celebrating my highs, and for providing your shoulder during my lows. Neeraj and Bhavi, thank you for all the gaming nights, weekend getaways, and for being the best flatmates.

Finally, I am forever grateful to my കുടുംബം, B.K Nair, Raji, Hema, Kumar, Vinod, and Jenny for their unwavering love, patience, understanding, and belief in me. A special thank you to my siblings, Jo and Teenasai, for being my mainstays. You have always been there for me through thick and thin and I am incredibly fortunate to have you two in my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Controlled thermonuclear fusion has long been hailed as the 'holy grail of energy generation' because it has the potential to provide clean and abundant energy for all our terrestrial needs. Although the concept of nuclear fusion has been known to us since the early 20^{th} century, the realization of a commercially viable fusion reactor has proven to be a great scientific and engineering challenge. Still, significant progress has been made in the last few decades, resulting in the construction of large-scale experimental reactors that are capable of achieving nuclear fusion.

Modern day fusion reactor concepts can be broadly categorized into two main types: Inertial Confinement Fusion (ICF) reactors, and Magnetic Confinement Fusion (MCF) reactors. An experimental MCF reactor that aims to deliver energy from controlled thermonuclear fusion and pave the way for commercially viable fusion is ITER [2] (International Thermonuclear Experimental Reactor). ITER is designed to perform fusion in a pulsed operation for eight minutes and if successful, will serve as the last step before fusion power plants are made. However, as power plants must run for longer time scales, there are still a number of challenges that must be addressed before fusion reactors are viable.

One such challenge has to do with the lifetime of the reactor wall components that are subjected to erosion and high heat loads. The erosion in the wall is primarily caused by edge plasma ions hitting the walls causing sputtering of the wall materials in a phenomena known as plasma-material interactions (PMI). The sputtered particles from the wall enter

the plasma, leading to the introduction of impurities that can negatively affect the overall performance of the reactor. To address these challenges, sheaths, the governing structures behind PMI must be studied.

This thesis provides an introductory overview of classical sheath theory and uses a 'continuum kinetic' approach to extend recent studies on the limitations of commonly used sheath models in cases where the sheath entrance is ill-defined. A recently proposed sheath model by Li et al. [1] that includes transport terms such as the heat flux, thermal force, and isotropization terms has been shown to resolve these limitations. The work performed and presented in this thesis considers and studies the Li et al. [1] model, and focuses on analyzing the performance of the continuum kinetic simulations in collisional sheaths. The continuum kinetic simulations are conducted by numerically solving the Vlasov-Maxwell equations using the discontinuous Galerkin method in the Gkeyll [3] software framework. The original results of this approach are benchmarked to the particle-in-cell results that were recently published to verify the new sheath model by Li et al. [1]. The results of the simulations offer insight into capabilities of the continuum kinetic approach and answers questions regarding the accuracy of the continuum kinetic approach.

Notes on Conventions Used This work follows the International System of Units (SI units) for all the quantities, barring the temperature, which is expressed in terms of units of energy. Specifically, Joules and electron-volts have been used. To express the temperature in terms of Joules, the temperature (in Kelvin) is multiplied with the Boltzmann constant ($k_B = 1.380649 \times 10^{-23} \frac{J}{K}$). The conversion from Joules to electron-volts can then be done by dividing the energy (in Joules) by the elementary charge constant ($e = 1.60217663 \times 10^{-19} C$). This work considers a Hydrogen plasma ($Z = 1$).

1.1 What is a Plasma?

The term 'plasma' was first coined by Irving Langmuir in 1928 when he used it to describe a region of ionized gas where ions and electrons were found in equal numbers [4]. Like ionized gases, plasmas are formed by providing a gas with enough energy to strip the electrons from its constituent atoms, resulting in a medium consisting of a large number of positively and negatively charged particles. Unlike ionized gases, however, plasmas are characterized by their 'quasi-neutrality'.

Quasi-neutrality is best explained by first understanding the concept of 'Debye shielding'. Debye shielding is the ability of a plasma to shield out applied electric potentials. To illustrate, let us consider a spherical anode in a plasma. Assuming that the anode is kept at a fixed potential, the electrons in the plasma will quickly form a spherical cloud around the anode. Due to the aggregation of the electrons, the potential of the anode in the plasma decreases spatially and approaches zero at the edge of the electron sphere, where the anode is completely shielded from the rest of the plasma. The effective distance over which this shielding occurs is given by the 'Debye length' [5]. The Debye length is mathematically expressed as,

$$\lambda_D = \sqrt{\frac{\epsilon_0 T_e}{n e^2}} \tag{1.1}$$

where ϵ_0 is the permittivity of free space, T_e is the electron temperature, n is the density of the plasma, and e is the elementary charge constant. Therefore, at large enough length scales i.e.„ length scales much larger than the Debye length, the charged particles within the plasma are shielded out and no large electric fields are formed within the bulk of the plasma. At this scale, the plasma is quasi-neutral.

Apart from quasi-neutrality, ionized gases must satisfy two other criteria to formally be considered a plasma [5]. The second criterion is that there must be enough charged particles in the plasma to perform Debye shielding successfully. This criterion is quantified by accounting for the number of particles in a 'Debye sphere', which is a sphere of a Debye length radius. The number of particles in a Debye Sphere is given by,

$$N_D = n\frac{4}{3}\pi\lambda_D^3 \tag{1.2}$$

therefore, the second criterion is satisfied when,

$$N_D \gg 1 \tag{1.3}$$

The third criterion is that the electron plasma frequency,

$$\omega_{pe} = \sqrt{\frac{n_e e^2}{m_e \epsilon_0}} \tag{1.4}$$

where n_e is the density of the electrons, m_e is the mass of the electron, must be greater than the electron-neutral collision frequency, ν_{en}. This criterion makes sure that the plasma is dominated by electromagnetic forces rather than kinetic forces. Hence,

$$\omega_{pe} > \nu_{en}. \tag{1.5}$$

1.2 The Plasma Sheath

Irving Langmuir was also the first to explore the concept of a 'sheath' when studying mercury arcs like those found in a mercury arc valve. He discovered that a negatively charged auxiliary electrode in the path of the arc would carry a current that seemed to be unaffected by the potential difference between the mercury cathode and the anode [6]. Langmuir explained this voltage independence by describing that the negative electrode repels the negative electrons and attracts the positive ions. The attracted ions then aggregate around the electrode and form a 'sheath', which is of a definite thickness and is composed purely of positive ions and neutral atoms. The sheath adjusts itself to keep the current in the electrode constant.

Plasma sheaths are formed when a material boundary comes in contact with a plasma, such as the auxiliary electrode in a mercury arc valve or the bounding walls of a tokamak reactor. In the case of a classical sheath where there are no emissions from the wall, the moment the plasma is in contact with the boundary, the electrons and ions recombine with the boundary and neutralize. In this way, the boundary acts as a sink, where the particles in the plasma are lost. This recombination does not occur simultaneously because the electrons possess a much higher thermal velocity than the ions, due to their significantly lower mass. The thermal velocity of the ions and electrons is expressed as,

$$V_{Te} = \sqrt{\frac{T_e}{m_e}} \tag{1.6}$$

$$V_{Ti} = \sqrt{\frac{T_i}{m_i}} \tag{1.7}$$

where T_i is the temperature of the ions and m_i is the mass of the ion. The difference in loss rates results in the boundary gaining a negative charge, repelling electrons, and attracting ions. The attracted ions develop a potential, called the floating potential [7] that does not

propagate through the bulk of the plasma due to Debye shielding. The floating potential then equalizes the electron and ion flux to the boundary and the net current at the boundary is zero. Hence, a dynamic equilibrium is reached [8] and a layer of net positive charge at the interface of the boundary and the plasma is established. This layer is called the plasma sheath. The thickness of the sheath is adjusted to maintain the dynamic equilibrium.

1.2.1 The Bohm Speed

Suppose we consider a section of a bounded plasma near a boundary. As we have seen above, a plasma sheath will form near the boundary, and ions in the immediate vicinity of the sheath will be attracted to the negatively charged boundary. Let us assume that the ions enter the sheath with an initial bulk velocity, $u_{i,0}$

Using the conservation of mass and the conservation of energy, we note that,

$$n_0 \mathbf{u}_{i,0} = n_i(x)\mathbf{u}_i(x) \tag{1.8}$$

$$\frac{1}{2} m_i u_{i,0}^2 = \frac{1}{2} m_i u_i^2(x) - e\phi(x). \tag{1.9}$$

Solving for the ion density,

$$n_i(x) = n_0 \left(1 - \frac{2e\phi(x)}{m_i u_{i,0}^2}\right)^{-\frac{1}{2}}. \tag{1.10}$$

The electrons in the plasma are considered to follow the Boltzmann relation [5] and so the density of electrons is given by,

$$n_e(x) = n_0 \exp\left(\frac{e\phi(x)}{T_e}\right). \tag{1.11}$$

Poisson's equation, which relates the electrostatic potential to the charge density is given by,

$$\nabla^2\phi = -\frac{\rho_c}{\epsilon_0} \tag{1.12}$$

or

$$\nabla^2\phi = \frac{e}{\epsilon_0}(n_e - n_i). \tag{1.13}$$

Using our results for the electron and ion densities in Poisson's equation,

$$\nabla^2\phi = \frac{e}{\epsilon_0}\left(n_0 \exp\left[\frac{e\phi(x)}{T_e}\right] - n_0\left[1 - \frac{2e\phi(x)}{m_i u_{i,0}^2}\right]^{-\frac{1}{2}}\right) \tag{1.14}$$

$$\nabla^2\phi = \frac{n_0 e}{\epsilon_0}\left(\exp\left[\frac{e\phi(x)}{T_e}\right] - \left[1 - \frac{2e\phi(x)}{m_i u_{i,0}^2}\right]^{-\frac{1}{2}}\right). \tag{1.15}$$

Simplifying the above expression in terms of the following variables,

$$\chi = -\frac{e\phi(x)}{T_e} \tag{1.16}$$

$$\xi = x\sqrt{\frac{n_0 e^2}{\epsilon_0 T_e}} \tag{1.17}$$

$$M = \frac{u_0}{\sqrt{\frac{T_e}{m_i}}}. \tag{1.18}$$

We get,

$$\frac{d^2\chi}{d\xi^2} = \left(1 + \frac{2\chi}{M^2}\right)^{-\frac{1}{2}} - \exp(-\chi). \tag{1.19}$$

Multiplying equation 1.19 with $\frac{d\chi}{d\xi}$, we get

$$\frac{d\chi}{d\xi}\left(\frac{d^2\chi}{d\xi^2}\right) = \frac{d\chi}{d\xi}\left(1 + \frac{2\chi}{M^2}\right)^{-\frac{1}{2}} - \frac{d\chi}{d\xi}\exp(-\chi). \tag{1.20}$$

We can now integrate equation 1.20 with respect to ξ', a dummy variable to find χ.

$$\int_0^\xi \frac{d\chi}{d\xi'}\left(\frac{d\chi}{d\xi'^2}\right)d\xi' = \int_0^\xi \left(\frac{d\chi}{d\xi'}\right)\frac{1}{\sqrt{1+\frac{2\chi}{M^2}}}d\xi' - \int_0^\xi \left(\frac{d\chi}{d\xi'}\exp(-\chi)\right)d\xi'. \tag{1.21}$$

Consider we are at the point $x = 0$. At this point we can deduce that $\xi = 0$. In a physical sense, $x = 0$ is the boundary between the bulk plasma and the sheath. At this point, we also deduce that the plasma is quasi-neutral and take $\chi = 0$. Without the presence of an electrostatic potential, an electric field cannot develop, and hence,

$$\left.\frac{d\chi}{d\xi'}\right|_{\xi'=0} = 0. \tag{1.22}$$

Using this result in equation 1.21 we get,

$$\frac{1}{2}\left(\frac{d\chi}{d\xi'}\right)^2\bigg|_{\xi} = M^2\left(\sqrt{1+\frac{2\chi}{M^2}}-1\right)+\left(\exp(-\chi)-1\right). \tag{1.23}$$

In the above equation, the left-hand side term is raised to two, an even number. Therefore, the right-hand side term of the equation must always be positive. We use this observation to expand the right-hand side term of the above equation at $\chi \ll 1$ using the Taylor series expansion, as the above integral cannot be evaluated analytically [5].

$$M^2\left[1+\frac{\chi}{M^2}-\frac{1}{2}\frac{\chi^2}{M^4}+\cdots-1\right]+1-\chi+\frac{1}{2}\chi^2+\cdots-1\geq 0 \tag{1.24}$$

$$\frac{1}{2}\chi^2\left(1-\frac{1}{M^2}\right)\geq 0. \tag{1.25}$$

This inequality is only satisfied when,

$$\left(1-\frac{1}{M^2}\right)\geq 0 \tag{1.26}$$

or,

$$\frac{m_i u_0^2}{T_e}\geq 0 \tag{1.27}$$

thus,

$$u_0 \geq u_B = \sqrt{\frac{T_e}{m_i}}. \tag{1.28}$$

Where u_B is defined as the 'Bohm speed'. The inequality derived above is known as the

Bohm sheath criterion. This criterion specifies that for a sheath to form, the ions must reach a velocity equal to or higher than the Bohm speed. To understand why, let us consider the motion of the ions near the sheath. As the boundary is at a lower potential than the bulk plasma, ions will accelerate under the presence of the electric field toward the boundary. This acceleration results in an increase of the ion velocity and as per the law of conservation of mass, a decrease in the density. If the ions entering the sheath are slower than the Bohm speed, they will accelerate sharply and cause the ion density to drop below the electron density. In this case, the polarity of the potential is flipped, and Debye shielding cannot occur. Mathematically, we can interpret that if the Bohm criterion is not satisfied, oscillatory solutions are produced for the potential inside the sheath [9]. Therefore, the criterion provides a necessary condition for the development of a stable, well-behaved sheath.

1.3 Sheath Models

Since the early works of Langmuir and Bohm, sheaths have been studied extensively [10] and various, more encompassing formulations for the Bohm speed have been described [11].

For example, while the applicability of Bohm's original formulation is limited due to the inherent cold ions assumption, more recent adiabatic fluid models [12] and isothermal fluid models [13] factor in the finite ion temperatures, which are usually comparable to the electron temperature. Furthermore, many current plasma fluid solvers formulate the Bohm speed using the electron and ion heat capacity ratios [11]. These models are flexible in the treatment of ions, which can be considered cold, adiabatic, or isothermal. The Bohm speed using the heat capacity ratios is,

$$u_B = \sqrt{\frac{\gamma_i T_i + \gamma_e T_e}{m_i}}. \tag{1.29}$$

Another approach to formulating a more generally applicable Bohm speed is by using the kinetic theory of plasma that allows a more unconstrained treatment of the ions and electrons [14]. The sheath criterion according to kinetic theory takes the form,

$$\left.\frac{\partial n_e}{\partial \phi}\right|_{\phi=0} \geq -\frac{e}{m_i}\int_0^\infty \frac{1}{v^2} f_i dv. \tag{1.30}$$

Assuming the electrons to follow the Boltzmann relation, we get the more familiar,

$$-\frac{q_e}{T_e} n_e \geq -\frac{q_e n_i}{m_i u_i^2} \tag{1.31}$$

$$u_i^2 \geq \frac{T_e}{m_i}\frac{n_i}{n_e}. \tag{1.32}$$

Now, if a two-scale system is considered [9], there exists a point that divides the plasma into a quasi-neutral bulk plasma regime and a non-neutral sheath. This point is called the sheath entrance or the sheath edge. Here, the plasma is quasi-neutral, and the kinetic description simplifies into,

$$u_i^2 \geq \frac{T_e}{m_i} \tag{1.33}$$

which is the original formulation derived by Bohm. We can also retrieve the sheath criterion using the two-scale system by linearizing Poisson's equation near the sheath entrance,

$$\frac{\partial^2 \phi}{\partial x^2} = 4\pi e\phi \left(\frac{\partial n_e}{\partial \phi} - \frac{\partial n_i}{\partial \phi} \right) \Bigg|_{se} \tag{1.34}$$

For a non-oscillatory solution,

$$\left(\frac{\partial n_e}{\partial \phi} - \frac{\partial n_i}{\partial \phi} \right) \Bigg|_{se} \geq 0. \tag{1.35}$$

Here, the subscript *se* denotes the sheath entrance. We can further derive the Bohm speed from this inequality by also considering the plasma conservation equations, the result of which would be the same as the Bohm speed derived from the fluid model [11].

1.3.1 Limitations of Current Sheath Models

As mentioned above, the two-scale model which was used to derive the sheath criterion and Bohm speed defines the sheath entrance as a distinct point where the plasma is divided into the sheath and the bulk plasma. This model, however, is only applicable in the asymptotic limit [15],

$$\frac{\lambda_D}{L} \to 0. \tag{1.36}$$

While this limitation may not seem significant, it has been found that in many numerical and experimental applications [16], the asymptotic limit condition is not satisfied and consequently a clear sheath entrance cannot be established.

Instead, by considering Boltzmann electrons and cold ions in a matched asymptotic analysis [17], an intermediate 'sheath-transition region' is found to exist. This region, which exists between the bulk plasma and the sheath, is on the order of several Debye lengths and weakly

violates quasi-neutrality. The assumption of equation 1.36 is not limited only to the two-scale analysis, and the aforementioned fluid and kinetic models also inherently assume this asymptotic case.

Also, recent first-principle kinetic simulations by Tang [18] show that the assumption of isothermal and isotropic electrons in classical sheath theory is not valid. Particle recombination at the boundaries introduce significant temperature gradients for ions and electrons and a strong temperature anisotropy exists for electrons. Hence the kinetic, fluid and two-scale models are inaccurate for most cases of interest, i.e., cases where the sheath entrance is not well defined. We refer to this problem as the sheath-transition problem.

Moreover, when the effect of collisions on the Bohm speed is considered, there is a stark difference between the predictions from the fluid model and the findings from kinetic simulations [19]. Highly collisional plasma have low heat capacity ratios due to the frequent inter-species and intra-species collisions that help isotropize the plasma. Consistently, high heat capacity ratios are found in plasma with low collisional frequencies. Looking at equation 1.29, the fluid model predicts an inversely proportional relationship between the Bohm speed and the collisionality i.e., as the collisionality increases, the heat capacities decrease and by extension, the Bohm speed decreases. Kinetic simulations from Tang and Guo [19], however, report the opposite. They find that as the collisionality of the plasma increases, so does the Bohm speed.

This inconsistency is resolved by Tang and Guo in the same work, and they achieve this by including the electron heat flux, an energy transport term in their Bohm speed formulation. Additionally, they point out that plasma transport, which is prevalent in the sheath region is often overlooked in the fluid sheath models. Identifying the important role of transport physics, Li et al. [1] follow a similar approach and consider terms from the transport equations in a novel sheath model to resolve the previously mentioned sheath-transition problem.

In the next chapter, we look at what these transport terms are and examine their role in the sheath-transition problem. Li's transport inclusive modified Bohm speed formulation is also derived. A brief overview of the continuum kinetics approach is given.

Chapter 2

Methodology

The inconsistency between fluid model predictions and kinetic simulation results on the variation of the Bohm speed with collisionality has been resolved by Tang and Guo [19]. They include the parallel electron heat flux, the dominant energy transport term, instead of the isothermal assumption for electrons to arrive at the following Bohm speed equation,

$$u_i \geq \sqrt{\frac{3\left(\beta T_{e,\|} + T_{i,\|}\right)}{m_i}} \tag{2.1}$$

where β, the heat flux factor is defined as,

$$\beta = \frac{1 - \frac{\partial}{\partial \phi}\left(\frac{q_n^i}{e n_i^{se} u_{i,\|}^{se}}\right)}{1 - \frac{\partial}{\partial \phi}\left(\frac{q_n^e}{e n_e^{se} u_{e,\|}^{se}}\right)}. \tag{2.2}$$

Here, $T_{e,\|}$ and $T_{i,\|}$ are the electron and ion temperatures parallel to the flow direction of the plasma. q_n^e and q_n^i are the electron and ion heat fluxes. n_e^{se} and n_i^{se} are the electron and ion densities at the sheath entrance, and $u_{e,\|}^{se}$, $u_{i,\|}^{se}$ are the electron and ion drift velocities at the sheath entrance respectively.

It is worth noting that this model requires the evaluation of the density and flow velocity at the sheath entrance, which as we have previously mentioned, is clearly defined only in cases within the asymptotic limit. For cases away from this limit, there exists an ambiguity

regarding the sheath entrance. Therefore, we now look at Li's[1] model that defines the sheath entrance in cases where a sheath-transition region is found to exist.

2.1 Plasma Transport Equations and the Bohm speed

Li [1] begins to tackle the sheath-transition problem by first considering collisional plasma in the regime of $K_n^{sh} > 1$ to $K_n^{sh} \gg 1$, as transport is significant in this regime. Here, K_n^{sh} is the sheath Knudsen number, a dimensionless quantity that helps quantify the collisionality of the sheath. It is defined as,

$$K_n^{sh} = \frac{\lambda_{mfp}}{\lambda_D} \tag{2.3}$$

where λ_{mfp} is the mean free path and λ_D is the Debye Length. The mean-free path is the distance traveled by a particle before it undergoes a Coulomb collision. Therefore, highly collisional plasma have relatively shorter mean free paths and lower sheath Knudsen numbers than a low collisionality plasma. The mean free path can be estimated using,

$$\lambda_{mfp} = \frac{v_{T,e}}{\nu_{ei}} \tag{2.4}$$

where $v_{T,e}$ is the electron thermal velocity and ν_{ei} is the anisotropic electron-ion collision frequency. The anisotropic formulation is used because of the strong temperature anisotropy in the Knudsen layer [18], defined as one λ_{mfp} from the boundary. The anisotropic electron-ion collision frequency is defined as [20],

$$\nu_{ei} = \frac{\sqrt{\pi}}{2} n_i \frac{e^4}{(4\pi\epsilon_0)} \frac{\ln\Lambda}{\sqrt{m_e T_{ex}} T_{ey}} \tag{2.5}$$

where $\ln\Lambda$ is the coulomb logarithm, which is typically in the range of 5 to 15. In the collisional regime of interest i.e., $\left(K_n^{sh} > 1 \rightarrow K_n^{sh} \gg 1\right)$, collisional forces that primarily affect the momentum transport in the plasma consist of the thermal force and the friction force [21]. Out of these, the friction force, which arises due to the differences in the electron and ion flow velocities becomes negligible when the plasma reaches a steady state, as the electron and ion flux equalize when the sheath is developed. The thermal force that arises due to temperature gradients in the plasma, however, is prevalent because the electrons are not isothermal.

The energy equation, on the other hand, is primarily affected by the heat flux and the intra-species collisions (electron-electron and ion-ion collisions) that arise due to the significant temperature anisotropy in both electrons and ions. These collisions drive the thermal isotropization of each species. While inter-species collisions (electron-ion and ion-electron) are also present between the ions and electrons, they do not influence the energy equation as much because of the large ion-electron mass ratio. Ohmic heating can also be an influencing factor in the energy equation, but as the plasma reaches a steady state, the net current becomes negligible and the effect of Ohmic heating diminishes.

Following the work of Li et al. [1], we can now derive the transport equations by taking the moments of the Boltzmann equation (see subsection 2.2.2) and by including the thermal force, electron heat flux, and temperature isotropization terms. We neglect any particle source terms as we assume that any particle addition takes place in the bulk plasma. The transport equations are[1],

$$\frac{\partial(n_e u_{ex})}{\partial x} = 0 \tag{2.6}$$

$$\frac{\partial(n_i u_{ix})}{\partial x} = 0 \tag{2.7}$$

$$n_e m_e u_{ex} \frac{\partial u_e x}{\partial x} + \frac{\partial(n_e T_{ex})}{\partial x} = e n_e \frac{\partial \phi}{\partial x} - \alpha n_e \frac{dT_{ex}}{dx} \tag{2.8}$$

$$n_i m_i u_{ix} \frac{\partial u_i x}{\partial x} + \frac{\partial(n_i T_{ix})}{\partial x} = -e n_i \frac{\partial \phi}{\partial x} - \alpha n_i \frac{dT_{ix}}{dx} \tag{2.9}$$

$$n_e u_{ex} \frac{\partial T_{ex}}{\partial x} + 2 n_e T_{ex} \frac{\partial u_{ex}}{\partial x} + \frac{\partial q_n^e}{\partial x} = Q_{ee} + Q_{ei} \tag{2.10}$$

$$n_i u_{ix} \frac{\partial T_{ix}}{\partial x} + 2 n_i T_{ix} \frac{\partial u_{ix}}{\partial x} + \frac{\partial q_n^i}{\partial x} = Q_{ii} + Q_{ie}. \tag{2.11}$$

Here, q_n^e and q_n^i are the electron and ion heat fluxes.Q_{ei}, Q_{ee}, Q_{ii}, and Q_{ie} are the electron-electron, electron-ion, ion-ion, and ion-electron temperature isotropization factors respectively. These factors quantify the degree by which the temperature of the plasma isotropizes with inter-species and intra-species collisions. Note that x in the transport equations refers to the direction parallel to the axis of the boundary, similar to the parallel direction in Tang and Guo's formulation.

We also note that the thermal force in the momentum equation is defined as

$$R_T = -\alpha n_e \frac{dT_{ex}}{dx} \tag{2.12}$$

where α is the thermal force coefficient. To avoid assuming the heat capacity ratios, γ_e and

γ_i, the transport equations are closed by evaluating higher order moments for the heat flux (q_n^e, q_n^i), and using simulation data for the collisional terms (Q_{ei}, Q_{ee}, Q_{ii}, Q_{ie}, R_T).

With the help of the transport equations, we can now derive an expression for the Bohm speed that is inclusive of the thermal force, heat fluxes, and isotropization terms. To do this we first combine the continuity, momentum, and energy conservation equations to find the change of the species density with respect to the potential. Using the electron equations and neglecting the electron mass,

$$\frac{\partial n_e}{\partial \phi} = \frac{e n_e}{(3+2\alpha)T_{ex}} + \frac{1+\alpha}{(3+2\alpha)u_{ex}T_{ex}}\left(\frac{\partial q_n^e}{\partial \phi} + \frac{Q_{ee}+Q_{ei}}{E}\right) \tag{2.13}$$

for electrons, and

$$\frac{\partial n_i}{\partial \phi} = \frac{1}{3u_i T_{ix} - m_i u_{ix}^3}\left(\frac{\partial q_n^i}{\partial \phi} + \frac{Q_{ii}+Q_{ie}}{E}\right) - \frac{e n_i - \alpha n_e \frac{\partial T_{ex}}{\partial \phi}}{3T_i - m_i u_{ix}^2} \tag{2.14}$$

for ions. We can now use this in the Bohm sheath criterion,

$$\frac{\partial n_e}{\partial \phi} - \left.\frac{\partial n_i}{\partial \phi}\right|_{se} \geq 0 \tag{2.15}$$

which upon substitution becomes,

$$\begin{aligned} &\left\{\frac{e n_e}{(3+2\alpha)T_{ex}} + \frac{1+\alpha}{(3+2\alpha)u_{ex}T_{ex}}\left(\frac{\partial q_n^e}{\partial \phi} + \frac{Q_{ee}+Q_{ei}}{E}\right)\right\} \\ &-\left\{\frac{1}{3u_i T_{ix} - m_i u_{ix}^3}\left(\frac{\partial q_n^i}{\partial \phi} + \frac{Q_{ii}+Q_{ie}}{E}\right) - \frac{e n_i - \alpha n_e \frac{\partial T_{ex}}{\partial \phi}}{3T_i - m_i u_{ix}^2}\right\}\Bigg|_{se} \geq 0 \end{aligned} \tag{2.16}$$

from which the Bohm speed as derived by Li et al. [1] is found to be,

$$u_B = \sqrt{\frac{(\beta T_{ex}^{se} + 3T_{ix}^{se})}{m_i}} \tag{2.17}$$

where β, according to Li et al. [1] is defined as,

$$\beta = \frac{3 - \left[\frac{3+2\alpha}{en_i^{se}u_{ix}^{se}}\left(\frac{\partial q_n^i}{\partial \phi} + \frac{Q_{ii}+Q_{ie}}{E}\right)\right] + \left[\frac{\alpha}{en_e^{se}u_{ex}^{se}}\left(\frac{\partial q_n^e}{\partial \phi} + \frac{Q_{ee}+Q_{ei}}{E}\right)\right]}{1 + \left[\frac{1+\alpha}{en_e^{se}u_{ex}^{se}}\left(\frac{\partial q_n^e}{\partial \phi} + \frac{Q_{ee}+Q_{ei}}{E}\right)\right]} \tag{2.18}$$

To compute β, we evaluate the transport terms as follows.

Firstly, the thermal force, which is evaluated using,

$$R_T = \iiint m_e v_x \left(\frac{\partial f}{\partial t}\right)_{\text{coll}}^{ei} d^3v \tag{2.19}$$

from which α can be found using,

$$\alpha = -\frac{R_T}{n_e\left(\frac{dT_{ex}}{dx}\right)}. \tag{2.20}$$

Secondly, the heat flux is found using the integrals as described by Chew et al. [22]. The electron heat flux is found using

$$q_n^e = \iiint m_e (v_x - u_x)^3 \, f \, d^3v \tag{2.21}$$

and the ion heat flux is found using

$$q_n^i = \iiint m_i (v_x - u_x)^3 \, f \, d^3v. \tag{2.22}$$

Finally, the isotropization terms are evaluated using the following expression,

$$Q_{sr} = \iiint m_e(v_x - u_x)^2 \left(\frac{\partial f}{\partial t}\right)^{sr}_{\text{coll}} d^3v. \tag{2.23}$$

Here, $\left(\frac{\partial f}{\partial t}\right)^{sr}_{\text{coll}}$ is the change in the distribution function due to collisions. The superscript sr stands for the species in question. (e.g.) $\left(\frac{\partial f}{\partial t}\right)^{ei}_{\text{coll}}$ is the change in the electron distribution function due to electron-ion collisions.

2.2 Kinetic Plasma Equations

Li et al. [1] in their work verified the accuracy of their Bohm speed formulation by conducting numerical simulations. Particularly, they conducted particle-in-cell simulations to capture the kinetic behavior in the plasma. Simulation has established itself as an important part of scientific analysis, alongside theory and experiment as they allow us to mathematically model and analyze complex systems that may not have an analytical solution.

At face value, developing a complete plasma model appears to be straightforward. The equations of motion of a charged particle in an electromagnetic field are well established and so are the numerical methods to solve these equations. Therefore, we can, in theory, load up the necessary charged particles in a domain and track their evolution by computing the Coulomb force experienced by each of the particles. This type of model is called an N-body model.

However, when the number of particles in a typical plasma, which is on the order of 10^{20}, is accounted for, we can see how unfeasible such a model can become. It has been shown [23] that if this type of model is considered, the computational complexity is $O(n^2)$. This means that doubling the number of particles leads to a quadruple increase in computation time.

For example, a typical simulation using this model with 10^{23} particles running for a time of 10^3 seconds demands approximately 10^{60} floating point operations per time step. If we run this simulation on a current supercomputer that can conduct 10^{18} floating point operations per second, each time step would take 10^{42} seconds, which is about 10^{25} times longer than the age of the universe!

Thankfully, it is not necessary to track every single particle of the plasma to get an accurate representation of its behavior. As detailed below, feasible mathematical models have been derived and numerical solutions to these models have been attained in a reasonable amount of time. Plasma models mainly fall into two broad categories, kinetic models and fluid models. Kinetic models are generally expensive to compute but capture much of the discrete plasma phenomenon, while fluid models are computationally cheaper but are only capable of capturing the more bulk plasma phenomena. To verify the modified Bohm speed formulation, we are interested in the kinetic description of the plasma as we would like to capture the important kinetic features of the sheath in the sheath-transition problem.

2.2.1 Klimontovich equation

We begin to kinetically model a plasma by tracking the motion of a single particle in three-dimensional physical space and velocity space. The density of this particle can be expressed as ,

$$N(\mathbf{x}, \mathbf{v}, t) = \delta(\mathbf{x} - \mathbf{X}(t))\delta(\mathbf{v} - \mathbf{V}(t)) \tag{2.24}$$

where δ is the Dirac delta function and $\mathbf{X}, \mathbf{V}$ are the position and velocity vectors of the particle respectively. Therefore, if we were at a point $\mathbf{x}$ in configuration space i.e., the normal

physical space, and $\mathbf{v}$ in velocity space that coincided with the position and velocity vector of the particle, the Dirac delta functions would return a value of one and indicate the presence of a particle at that point. Now let us consider the case of a plasma consisting of two species with N particles each. The density of each specie 's' in this case can be expressed as,

$$N_s(\mathbf{x}, \mathbf{v}, t) = \sum_{i=1}^{N} \delta(\mathbf{x} - \mathbf{X}_i(t))\delta(\mathbf{v} - \mathbf{V}_i(t)). \tag{2.25}$$

The total density is then expressed as,

$$N(\mathbf{x}, \mathbf{v}, t) = N_i(\mathbf{x}, \mathbf{v}, t) + N_e(\mathbf{x}, \mathbf{v}, t). \tag{2.26}$$

Now that we know how the particles are distributed in physical and velocity space, we would like to find out how this distribution evolves with time. Let us first consider the effect of the Lorentz force on these particles. For a particle 'i' belonging to specie s,

$$m_s \frac{d\mathbf{V}_i}{dt}(t) = e\left(\mathbf{E}'(\mathbf{X}_i(t), t) + [\mathbf{V}_i(t) \times \mathbf{B}'(\mathbf{X}_i(t), t)]\right). \tag{2.27}$$

Where the electric field, $\mathbf{E}'(\mathbf{X}_i(t), t)$ and magnetic field, $\mathbf{B}'(\mathbf{X}_i(t), t)$ are the fields that arise self consistently due to the motion of the charged particles, excluding the particle in consideration itself. This expression gives us a description of the derivative of the velocity of the species. To find the derivative of the position of the species, we note that the velocity of the particles is the time derivative of the positions,

$$\mathbf{V}(t) = \frac{d\mathbf{X}(t)}{dt} \tag{2.28}$$

and for a particle, i, this becomes,

$$\mathbf{V}_i(t) = \frac{d\mathbf{X}_i(t)}{dt}. \tag{2.29}$$

Finally, we can also study the time evolution of the particles using our expression for the density by taking the time derivative,

$$\begin{aligned}\frac{\partial N_s(\mathbf{x},\mathbf{v},t)}{\partial t} = -\Bigg[&\sum_{i=1}^{N} \frac{\mathbf{X}_i(t)}{dt} \cdot \nabla_{\mathbf{x}}\big\{\delta(\mathbf{x}-\mathbf{X}_i(t))\delta(\mathbf{v}-\mathbf{V}_i(t))\big\} \\ &+ \sum_{i=1}^{N} \frac{d\mathbf{V}_i(t)}{dt} \cdot \nabla_{\mathbf{v}}\big\{\delta(\mathbf{x}-\mathbf{X}_i(t))\delta(\mathbf{v}-\mathbf{V}_i(t))\big\}\Bigg].\end{aligned} \tag{2.30}$$

Using our results for the derivative of position and velocity, we can obtain,

$$\begin{aligned}\frac{\partial N_s(\mathbf{x},\mathbf{v},t)}{\partial t} = -\Bigg[&\sum_{i=1}^{N} \mathbf{V}_i(t) \cdot \nabla_{\mathbf{x}}\big\{\delta(\mathbf{x}-\mathbf{X}_i(t))\delta(\mathbf{v}-\mathbf{V}_i(t))\big\} \\ &+ \sum_{i=1}^{N} \frac{e}{m_s}\big\{\mathbf{E}'(\mathbf{X}_i(t),t) + [\mathbf{V}_i(t) \times \mathbf{B}'(\mathbf{X}_i(t),t)]\big\} \cdot \nabla_{\mathbf{v}}\big\{\delta(\mathbf{x}-\mathbf{X}_i(t))\delta(\mathbf{v}-\mathbf{V}_i(t))\big\}\Bigg].\end{aligned} \tag{2.31}$$

Applying the identity, $a\delta(a-b) = b\delta(b-a)$, we can write equation 2.31 as,

$$\frac{\partial N_s(\mathbf{x},\mathbf{v},t)}{\partial t} = -\mathbf{v}\cdot\nabla_{\mathbf{x}}\sum_{i=1}^{N}\big\{\delta(\mathbf{x}-\mathbf{X}_i(t))\delta(\mathbf{v}-\mathbf{V}_i(t))\big\} - \frac{e}{m_s}\big\{\mathbf{E}'(\mathbf{x},t)+\big[\mathbf{v}\times\mathbf{B}'(\mathbf{x},t)\big]\big\}\cdot\nabla_{\mathbf{v}}\sum_{i=1}^{N}\big\{\delta(\mathbf{x}-\mathbf{X}_i(t))\delta(\mathbf{v}-\mathbf{V}_i(t))\big\}. \tag{2.32}$$

Using our initial definition for the specie density i.e., equation 2.25, we can obtain,

$$\frac{\partial N_s(\mathbf{x},\mathbf{v},t)}{\partial t} + \mathbf{v}\cdot\nabla_{\mathbf{x}}N_s(\mathbf{x},\mathbf{v},t) + \frac{e}{m_s}\left\{\mathbf{E}'(\mathbf{x},t)+[\mathbf{v}\times\mathbf{B}'(\mathbf{x},t)]\right\}\cdot\nabla_{\mathbf{v}}N_s(\mathbf{x},\mathbf{v},t) = 0 \tag{2.33}$$

or simply,

$$\frac{\partial N_s}{\partial t} + \mathbf{v}\cdot\nabla_{\mathbf{x}}N_s + \frac{e}{m_s}\left\{\mathbf{E}'+[\mathbf{v}\times\mathbf{B}']\right\}\cdot\nabla_{\mathbf{v}}N_s = 0. \tag{2.34}$$

Equation 2.34 is the Klimontovich equation [24]. It provides us with a complete and exact kinetic description of a plasma. To close the Klimontovich equations we use Maxwell's equations,

$$\nabla\cdot\mathbf{E}'(\mathbf{x},t) = \frac{\rho'(\mathbf{x},t)}{\epsilon_0}. \tag{2.35}$$

$$\nabla\cdot\mathbf{B}'(\mathbf{x},t) = 0. \tag{2.36}$$

$$\nabla \times \mathbf{E}'(\mathbf{x}, t) = -\frac{\partial \mathbf{B}'(\mathbf{x}, t)}{\partial t}. \tag{2.37}$$

$$\nabla \times \mathbf{B}'(\mathbf{x}, t) = \mu_0 \left[\mathbf{j}'(\mathbf{x}, t) + \epsilon_0 \frac{\partial \mathbf{E}'(\mathbf{x}, t)}{\partial t} \right]. \tag{2.38}$$

Where,

$$\rho'(\mathbf{x}, t) = \sum_{i=1}^{N} e \int N_s(\mathbf{x}, \mathbf{v}, t)\, d\mathbf{v}. \tag{2.39}$$

$$\mathbf{j}'(\mathbf{x}, t) = \sum_{i=1}^{N} e \int \mathbf{v} N_s(\mathbf{x}, \mathbf{v}, t)\, d\mathbf{v}. \tag{2.40}$$

While the Klimontovich equation captures the individual particles and the discrete behavior to describe the plasma exactly, the equation is not practically usable due to the Dirac delta functions. However, the Klimontovich equation is a good starting point to obtain a kinetic model of the plasma.

2.2.2 Vlasov equation

Usually, instead of the exact solutions that contain detailed, individual characteristics of the plasma, we are interested in the average or approximate characteristics. For this reason, we now introduce the concept of the distribution function, which is defined as,

$$f_s(\mathbf{x}, \mathbf{v}, t) := \langle N_s(\mathbf{x}, \mathbf{v}, t) \rangle \tag{2.41}$$

where the angle brackets $\langle * \rangle$ denote the ensemble average in the context of statistical mechanics. The ensemble average is defined as the mean of a quantity over all possible micro-states, realizing the given macro-state. This distribution function tells us how many particles are present in the phase space volume $\Delta\mathbf{x}\Delta\mathbf{v}$ centered at $(\mathbf{x}, \mathbf{v})$ and the velocity of these particles range from $\mathbf{v}$ to $\mathbf{v} + \Delta\mathbf{v}$. As the particles gain and lose energy, they will enter or leave the phase space volume and this is reflected by fluctuations in the density, represented by

$$N_s(\mathbf{x}, \mathbf{v}, t) = f_s(\mathbf{x}, \mathbf{v}, t) + \delta N_s(\mathbf{x}, \mathbf{v}, t). \tag{2.42}$$

Similarly, the electric and magnetic fields are subject to fluctuations as well,

$$\mathbf{E}'(\mathbf{x}, \mathbf{v}, t) = \mathbf{E}(\mathbf{x}, \mathbf{v}, t) + \delta\mathbf{E}(\mathbf{x}, \mathbf{v}, t) \tag{2.43}$$

$$\mathbf{B}'(\mathbf{x}, \mathbf{v}, t) = \mathbf{B}(\mathbf{x}, \mathbf{v}, t) + \delta\mathbf{B}(\mathbf{x}, \mathbf{v}, t) \tag{2.44}$$

where,

$$\mathbf{E}(\mathbf{x}, \mathbf{v}, t) := \langle \mathbf{E}'(\mathbf{x}, \mathbf{v}, t) \rangle \tag{2.45}$$

$$\mathbf{B}(\mathbf{x}, \mathbf{v}, t) := \langle \mathbf{B}'(\mathbf{x}, \mathbf{v}, t) \rangle. \tag{2.46}$$

We also note that,

$$\langle \delta N_s \rangle = \langle \delta\mathbf{E} \rangle = \langle \delta\mathbf{B} \rangle = 0. \tag{2.47}$$

Using the ensemble average values obtained above, we can substitute them into the Klimontovich equation (equation 2.34) to derive the time evolution of the plasma,

$$\frac{\partial f_s}{\partial t} + \mathbf{v} \cdot \nabla_{\mathbf{x}} f_s + \frac{e}{m_s} (\mathbf{E} + \mathbf{v} \times \mathbf{B}) \cdot \nabla_{\mathbf{v}} f_s = -\frac{e}{m_s} \langle (\delta \mathbf{E} + \mathbf{v} \times \delta \mathbf{B}) \cdot \nabla_{\mathbf{v}} \delta N_s \rangle. \tag{2.48}$$

The right-hand side in the above equation consists of the ensemble average of the discrete quantities of the plasma and accounts for effects on the plasma due to collisions. Classifying them under the sum,

$$\sum_s \frac{\partial f_s}{\partial t} = -\frac{e}{m_s} \langle (\delta \mathbf{E} + \mathbf{v} \times \delta \mathbf{B}) \cdot \nabla_{\mathbf{v}} \delta N_s \rangle \tag{2.49}$$

we get the Boltzmann equation,

$$\frac{\partial f_s}{\partial t} + \mathbf{v} \cdot \nabla_{\mathbf{x}} f_s + \frac{e}{m_s} (\mathbf{E} + \mathbf{v} \times \mathbf{B}) \cdot \nabla_{\mathbf{v}} f_s = \sum_s \frac{\partial f_s}{\partial t}. \tag{2.50}$$

If the discrete collisional terms are neglected, we arrive at the Vlasov equation [25],

$$\frac{\partial f_s}{\partial t} + \mathbf{v} \cdot \nabla_{\mathbf{x}} f_s + \frac{e}{m_s} (\mathbf{E} + \mathbf{v} \times \mathbf{B}) \cdot \nabla_{\mathbf{v}} f_s = 0. \tag{2.51}$$

However, this equation only describes the time evolution of one species. When considering and modeling multiple species in the plasma, the Vlasov equation must be supported with additional equations to couple the species and evolve the self-consistent fields. For electrostatic cases, this is done using Poisson's equation,

$$\nabla^2 \phi = -\frac{\rho}{\epsilon_0}. \tag{2.52}$$

For electromagnetic cases, Maxwell's equations are used,

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}. \tag{2.53}$$

$$\nabla \times \mathbf{B} = \mu_0 \left(\mathbf{j} + \epsilon_0 \frac{\partial \mathbf{E}}{\partial t} \right). \tag{2.54}$$

Therefore, we can obtain the time evolution of the plasma species' positions and velocities by solving these governing equations (equations (2.51),(2.53),(2.54)). Note that equation 2.51 must be solved for each species in the plasma. The Vlasov equation is the fundamental plasma modeling equation and serves as the basis for the kinetic model and the fluid model. The equations of the fluid model are derived by taking moments of the Vlasov equation.

2.3 Continuum Kinetic Method

The Vlasov equation provides us with a statistical and kinetic description of a plasma species using the distribution function (f_s). Depending on whether the case is electrostatic or electromagnetic, the Vlasov equation can be closed with either Poisson's equation or Maxwell's equations. Since these equations are in the form of Partial Differential equations (PDEs), they can be solved using well-known numerical techniques. Solving these kinetic equations by approximating the PDE is known as the continuum kinetic method. This method is fundamentally different to the particle-in-cell method, which uses the Lorentz force interactions to simulate the plasma.

In this work we choose to solve the Vlasov-Maxwell equations in the Gkeyll simulation framework [3]. Gkeyll uses the discontinuous Galerkin (DG) method, which is a numerical method

classified under the finite element family of methods (FEM). DG is particularly advantageous because it combines features from FEM, such as the ability to manage complicated geometries and provide high order accuracy, and FVM, such as data locality (which makes it suitable for high performance computing) and flux limiters. Moreover, increasing the subcell accuracy is much cheaper to do in DG, as compared to other methods that use mesh refinement [26]. More importantly, DG allows us to capture the kinetic phenomena in the plasma with fluid model-like efficiency.

To get a top-level understanding of DG, we first look at the Continuous Galerkin or Galerkin method. The Galerkin and the discontinuous Galerkin method are classified under the 'methods of weighted residuals'[27], along with some other standard methods such as the Point Collocation method, the Subdomain Collocation method, and the Least Squares method. They are classified as such because of two reasons. Firstly, they represent the approximate solution of the PDE as a linear combination of the product of expansion coefficients and basis functions. Secondly, these methods minimize the error between the approximate and true solution by evaluating the integral of the residual times a method-specific weight function.

Let us illustrate what all of this means by taking an example. Consider the following PDE with the true solution, $y(x)$ in a one-dimensional domain of length L,

$$\frac{\partial y(x)}{\partial x} + f(x) = 0. \tag{2.55}$$

The approximate solution to this PDE, denoted by $\tilde{y}(x)$, formulated as the linear combination of the expansion coefficients and basis functions is,

$$\tilde{y}(x) = c_1\phi_1(x) + c_2\phi_2(x) + c_3\phi_3(x) + \ldots + c_N\phi_N(x) \tag{2.56}$$

which is,

$$\tilde{y}(x) = \sum_{i=1}^{N} c_i \phi_i(x). \tag{2.57}$$

Here, N refers to the number of terms we would like to include in our approximation, $\phi_i(x)$ are the basis functions and c_i are the unknown expansion coefficients that need to be solved for. The basis function is assumed to be of a certain form, for example, polynomial or trigonometric, and is chosen so that it satisfies the boundary conditions of the original PDE. The approximate solution can be made more accurate with the inclusion of more terms. Plugging this into our original PDE, we get,

$$\frac{\partial \tilde{y}(x)}{\partial x} + f(x) = R(x). \tag{2.58}$$

Since $\tilde{y}(x)$ is only an approximation, it cannot completely satisfy the PDE and we get a leftover 'residual error', $R(x)$. This residual error is the error between the true solution and the approximate solution. Hence to find the expansion coefficients such that the approximate solution satisfies the PDE to an acceptable error, we must drive the residual to zero. To do this, we define the integral

$$\int_0^L R(x) W(x) dx = 0. \tag{2.59}$$

Here, $W(x)$ is a weighting function and depends on the method of weighted residual being used. defining the weighting function as per the Galerkin method [28],

$$W(x) = \frac{\partial \tilde{y}(x)}{\partial c_i}. \tag{2.60}$$

Using our definition of the approximate solution,

$$W(x) = \frac{\partial}{\partial c_i}\left[\sum_{i=1}^{N} c_i \phi_i(x)\right] \tag{2.61}$$

which is,

$$W(x) = \phi_i(x). \tag{2.62}$$

Substituting this result in the error minimization integral,

$$\int_0^L R(x)W(x)dx = 0 \tag{2.63}$$

we get,

$$\int_0^L R(x)\phi_i(x)dx = 0. \tag{2.64}$$

This expression specifies that over the entire domain, the integral of the 'inner product' of the residual and the basis functions in function space must be zero. The inner product mentioned here is completely analogous to the dot product in vector space, and hence when the inner product between two functions is zero, it means that there is no component of one function in the direction of the other.

Therefore, the integral provides the condition that the orthogonal projection of the residual and the basis functions on each other must be zero to minimize the error between the true and approximate solution. This is called the orthogonality constraint. The orthogonality constraint is important because Galerkin, who the method is named after, found that when

the approximate solution is expressed in terms of the basis functions, the minimum residual error occurs when the residual error is orthogonal to the basis functions themselves [29]. Substituting our residual term in the integral, we get,

$$\int_0^L \left[\frac{\partial \tilde{y}(x)}{\partial x} + f(x)\right] \phi_i(x) dx = 0. \tag{2.65}$$

Using this we find the expansion coefficients that minimize the residual error.

When applying the Galerkin method in FEM, the basis functions are defined, and the expansion coefficients are found in each element. The weighted residual integral is then evaluated over the entire domain and the approximate global solution is found. In this way, the solution is continuous over the entire domain.

The discontinuous Galerkin is much like the Galerkin method as discussed above, with the difference that instead of evaluating the weighted residual integral over the entire domain, it is first evaluated in each cell to obtain the local solutions [30]. This allows DG to handle complicated geometries and discontinuities better than the Galerkin method. However, finding the local solution in each element also leads to multiplying defined solutions at the cell edges, but this is resolved using flux limiters as in the finite volume method.

In the Gkeyll simulation framework, The weighted residual integral to find the solution of the Vlasov equation in Gkeyll is reported by Cagas [31] as,

$$\int \left[\frac{\partial f_h^j(t, \mathbf{z})}{\partial t} + \nabla_{\mathbf{z}} \cdot (\alpha_h^j(\mathbf{z}) f_h^j(t, \mathbf{z}))\right] \psi_t(\mathbf{z}) d\mathbf{z} = 0. \tag{2.66}$$

Here, $\psi_t(\mathbf{z})$ stands for the basis function, and the term in the square bracket is the discretized Vlasov equation that represents the residual. It is expressed in terms of the phase space variable $\mathbf{z}$ which is the 6-dimensional combination of the 3 configuration space variables

$(\mathbf{x}, \mathbf{y}, \mathbf{z})$, and 3 velocity space variables $(\mathbf{v_x}, \mathbf{v_y}, \mathbf{v_z})$. The non-discretized Vlasov equation in phase space is,

$$\frac{\partial f}{\partial t} + \nabla_{\mathbf{z}} \cdot (\alpha \mathbf{f}) = 0 \tag{2.67}$$

where $\nabla_{\mathbf{z}}$ is defined as,

$$\nabla_{\mathbf{z}} = (\nabla_{\mathbf{x}}, \nabla_{\mathbf{v}}) \tag{2.68}$$

and α is defined as,

$$\alpha = \left(\mathbf{v}, \frac{e}{m}\left[\mathbf{E} + \mathbf{v} \times \mathbf{B}\right]\right). \tag{2.69}$$

The subscript h represents the approximate variable and j represents the index from the first cell to the last cell in phase space. Therefore f_h^j denotes the approximate distribution function in each cell. This integral is evaluated in each cell, for all the cells in the phase space.

2.4 Simulation Setup

The numerical simulations have been run on the Gkeyll 2.0 framework that uses the discontinuous Galerkin method to solve the Vlasov equation and perform the time evolution of the distribution functions. Specifically, the Vlasov-Maxwell equations are solved with SI unit inputs. In total, eleven cases have been simulated with different initial temperatures and densities so that a wide range of collisionality is covered. The simulations are initialized

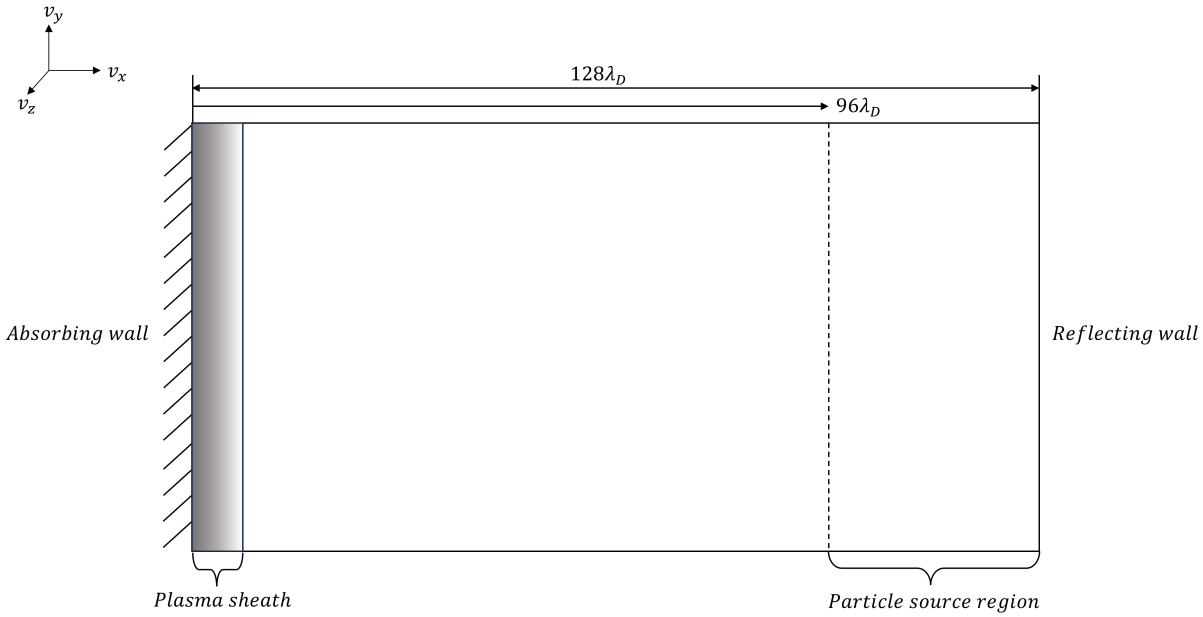


Figure 2.1: Illustration of the 1X3V simulation domain.

with temperatures,

$$T_i = T_e = T_0 \tag{2.70}$$

and densities,

$$n_i = n_e = n_0 \tag{2.71}$$

as a Maxwellian in a $1X3V$ (one spatial dimension, three velocity dimensions) domain. The electrons and ions are uniformly distributed in the plasma with zero initial flow velocity and are simulated until a steady state is achieved. The plasma is spatially bounded in a domain of length $128\lambda_D$, with the boundary at $0\lambda_D$ behaving as an absorbing wall and the boundary at $128\lambda_D$ behaving as a reflecting wall. The absorbing wall acts as a realistic material boundary and hence is where the plasma sheath forms.

As the sheath develops, the ions and electrons begin accelerating to the wall and are lost. To maintain the particle balance in the plasma, ions, and electrons are added into the domain far from the absorbing wall, at a distance of $96\lambda_D$ from it. The rate at which the ions and electrons are injected into the plasma is equal to the rate at which the ions exit the plasma. This is done so that the plasma equilibrates quickly. The velocity distribution of the injected ions and electrons are Maxwellian in nature and take the form,

$$f(v_x, v_y, v_z) = \frac{n}{\left(2v_T^2\pi\right)^{3/2}} \exp\left[-\frac{(v_x - u_x)^2 + (v_y - u_y)^2 + (v_z - u_z)^2}{2v_T^2}\right] \tag{2.72}$$

where the thermal velocity, v_T is evaluated at the initial temperature, T_0. Eleven cases have been simulated with various initial temperatures and densities. This is done to investigate the effect of collisions on the sheath-transition problem and the formulation of the Bohm speed since the collisional forces affect the momentum transport of the electrons. The collisionality of the plasma is characterized by the nominal Knudsen number, which is similar to the sheath Knudsen number except that the mean free path is defined using the isotropic collision frequency. The nominal Knudsen number is defined as,

$$K_n = \frac{\lambda_{\text{mfp}}}{\lambda_D} \tag{2.73}$$

where λ_D is the Debye length and λ_{mfp} is the mean free path. The mean free path in this case is expressed as,

$$\lambda_{\text{mfp}} = \frac{v_{T,e}}{\nu_{ei}}. \tag{2.74}$$

The electron thermal velocity $v_{T,e}$ is defined with the initial temperature,

$$v_{T,e} = \sqrt{\frac{T_0}{m_e}} \tag{2.75}$$

and the isotropic electron-ion collision frequency ν_{ei} is computed using the initial density and temperature [5],

$$v_{ei} = \frac{n_0 e^4}{\sqrt{m_e}(T_0)^{3/2}} \frac{\ln \Lambda}{16\pi\epsilon_0^2}. \tag{2.76}$$

The other isotropic collision frequencies can be calculated using the isotropic electron-ion collision frequency. The isotropic electron-ion collision frequency ν_{ee},

$$\nu_{ee} = \frac{v_{ei}}{\sqrt{2}}. \tag{2.77}$$

The isotropic ion-electron collision frequency ν_{ie},

$$\nu_{ie} = \frac{v_{ei}}{\sqrt{2}} \left(\frac{m_e}{m_i} \right). \tag{2.78}$$

The isotropic ion-ion collision frequency ν_{ii},

$$\nu_{ii} = \frac{v_{ei}}{\sqrt{2}} \left(\frac{v_{T,i}}{v_{T,e}} \right). \tag{2.79}$$

Therefore, by changing the initial temperature and density of the plasma, we can alter the nominal Knudsen number and hence change the collisionality of the plasma. A nominal Knudsen number range of twenty to five thousand has been considered. The initial temperatures and densities of the plasma for each case are given in table 2.1.

The Coulomb collisions have been implemented using the Lenard-Bernstein collisional model [32] as formulated in the Gkeyll framework [3]. This model computes the collisional integral (mentioned in section (2.2.2)) as,

$$\left(\frac{\partial f_s}{\partial t}\right)_c = \sum_r \nu_{sr} \frac{\partial}{\partial \mathbf{v}} \cdot \left[(\mathbf{v} - \mathbf{u}_{sr})\, f_s + v_{tsr}^2 \frac{\partial f_s}{\partial \mathbf{v}} \right]. \tag{2.80}$$

Where $\mathbf{u}_{sr}$ is the cross-species flow velocity, expressed as,

$$\mathbf{u}_{sr} = \mathbf{u}_s + \frac{\alpha_E}{2} \frac{m_s + m_r}{m_s n_s \nu_{sr}} (\mathbf{u}_r - \mathbf{u}_s) \tag{2.81}$$

$v_{T,sr}^2$ is the cross-species thermal velocity, expressed as,

$$v_{T,sr}^2 = v_{T,s}^2 + \frac{\alpha_E}{2} \frac{m_s + m_r}{m_s n_s \nu_{sr}} \left[\frac{1}{1 + \frac{m_s}{m_r}} \left(v_{T,r}^2 - \frac{m_s}{m_r} v_{T,s}^2 + \frac{1}{d_v} (\mathbf{u}_s - \mathbf{u}_r)^2 \right) \right] \tag{2.82}$$

α_E and δ_s are defined as,

$$\alpha_E = m_s n_s \nu_{sr} \delta_s \frac{1 + \beta}{m_s + m_r}. \tag{2.83}$$

$$\delta_s = \frac{2 m_s n_s \nu_{sr}}{m_s n_s \nu_{sr} + m_r n_r \nu_{rs}} \tag{2.84}$$

s and r denote the specie of interest and dv stands for the number of velocity dimensions. Another collision operator that approximates the collisional integral is the Bhatnagar-Gross-Krook operator (BGK). As per the Gkeyll implementation, the collisional derivative according to the BGK model is formulated as,

$$\left(\frac{\partial f_s}{\partial t}\right)_c = \sum_r \nu_{sr}\left(f_{Msr} - f_s\right) \tag{2.85}$$

where f_{Msr} is defined as,

$$f_{Msr} = \frac{n_s}{\left(2\pi v_{tsr}^2\right)^{d_v/2}} \exp\left[-\frac{\left(\mathbf{v} - \mathbf{u}_{sr}\right)^2}{2v_{tsr}^2}\right]. \tag{2.86}$$

The cross specie thermal velocity, v_{tsr}^2 and cross specie flow velocity $\mathbf{u}_{sr}$ is defined similarly to the LBO model,

$$v_{tsr}^2 = v_{ts}^2 - \frac{1}{d_v}\frac{\alpha_E}{m_s n_s \nu_{sr}}\left[d_v\left(m_s v_{ts}^2 - m_r v_{tr}^2\right) - m_r\left(\mathbf{u}_s - \mathbf{u}_r\right)^2 + 4\frac{\alpha_E}{m_s n_s \nu_{sr}}\left(m_s + m_r\right)^2\left(\mathbf{u}_s - \mathbf{u}_r\right)^2\right] \tag{2.87}$$

$$\mathbf{u}_{sr} = \mathbf{u}_s - \frac{\alpha_E}{2}\frac{m_s + m_r}{m_s n_s \nu_{sr}}\left(\mathbf{u}_s - \mathbf{u}_r\right) \tag{2.88}$$

.

The reason as to why the BGK model has been discussed will be apparent when the results of the continuum kinetic simulations are discussed. An example simulation input file for the case of $K_n = 50$ has been included in the appendix (A) for reference.

Case	Initial Temperature (eV)	Initial Density ($1/m^3$)	Nominal Knudsen Number
1	0.5	7.0×10^{20}	20
2	0.6	4.4×10^{20}	35
3	0.6	3.6×10^{20}	40
4	1.0	8.8×10^{20}	50
5	2.0	4.4×10^{20}	200
6	2.0	2.0×10^{20}	300
7	4.0	5.7×10^{20}	500
8	5.0	4.4×10^{20}	800
9	5.0	2.8×10^{20}	1000
10	2.6	1.0×10^{19}	2000
11	6.0	2.0×10^{19}	5000

Table 2.1: Initial temperature and density inputs for corresponding nominal Knudsen numbers

Chapter 3

Results and Analysis

3.1 The Sheath Entrance in the Sheath-Transition Region

The modified Bohm speed (2.17) and the dependant β term (2.18) is evaluated using the transport terms at the sheath entrance. For the cases in consideration that are away from the asymptotic limit, this sheath entrance is ambiguous due to the presence of the sheath-transition region. To evaluate the modified Bohm speed, we must first verify the existence of the sheath-transition region and then establish the sheath entrance in these cases. This chapter verifies the existence of the sheath-transition region and establishes the sheath entrance from the results of the continuum kinetic simulations. Further, the continuum kinetic results are compared with the theory and particle-in-cell results from Li et al. [1].

3.1.1 Existence of the Sheath-transition Region

The existence of the sheath-transition region can be verified by looking at the 'sheath profiles'. These profiles track the behavior of the macroscopic plasma properties near the sheath and provide insight into the development of the sheath and if present, the sheath-transition layer.

The sheath profiles that we have chosen to show the existence of the sheath-transition layer

are the the ion and electron densities, ion and electron temperatures, plasma potential, and the ion flow speed. All the profiles are plotted in the region of zero Debye lengths to eighty Debye lengths. This specific region is chosen away from where the particles are injected into the plasma ($96\lambda_D$).

The first plasma profile we plot in figure (3.1) is the ion and electron density profile for Knudsen numbers $K_n = 50$, $K_n = 500$, and $K_n = 2000$. The densities are normalized to the initial density of the species (n_0). These cases represent a highly collisional plasma, a moderately collisional plasma and a weakly collisional plasma.

It is evident across all collisionalities that there is a gradual decline in the ion and electron densities up to a distance of less that ten Debye lengths from the wall, where the decrease is sharp. This sharp decrease is indicative of the plasma sheath and this gradual decline in particle density reflects the presence of the sheath-transition region.

As the collisionality decreases, the gradient of the density over the distance also decreases. In the $K_n = 50$ case, the density ratio at which the decline becomes sharp is approximately 0.6, while for the case of $K_n = 2000$, this point is closer to 0.8.

The presence of the sheath-transition region is clearly noticed when the potential profiles are plotted as in figure (3.2). The potential in these plots are normalized to the initial electron temperature divided by the charge of the electron ($\frac{T_{e0}}{e}$). We can see that the potentials across all the cases decline gradually, similar to how the densities declined over the domain. This gradual decrease represents a weak violation of quasi-neutrality and the presence of a sheath-transition region. The strong violation of quasi-neutrality in the region less than 10 Debye lengths is where the sheath forms.

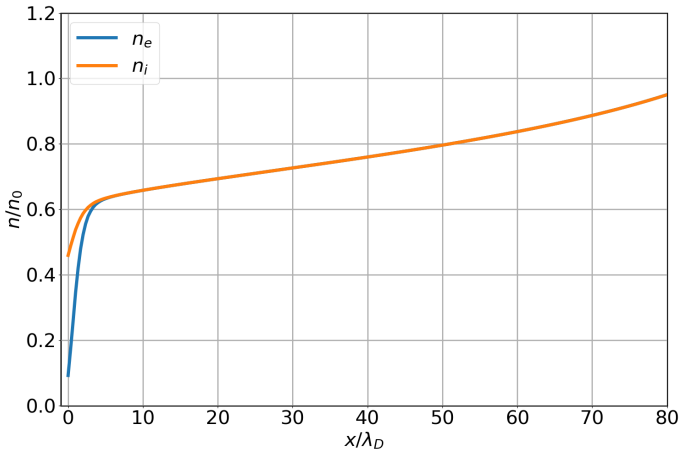
The potential profiles also display the same trend of a decreasing gradient with decrease in collisionality, suggesting that as the collisionality decreases, the sheath-transition region

begins to disappear.

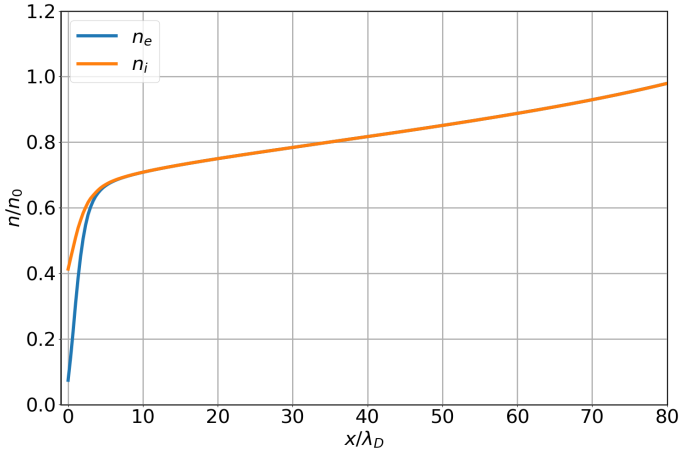
Finally, the ion speed profiles, normalized to the sound speed $\left(\sqrt{\frac{(T_{ex}+3T_{ix})}{m_i}}\right)$ in figure (3.3) also illustrate the existence of the sheath-transition region, in agreement with the density and potential profiles. Within the transition region, the ions accelerate gradually and when in the sheath, accelerate sharply. This sharp acceleration is predicted by the classical sheath theory and confirms the formation of the sheath in the region of zero to ten Debye lengths.

Figure (3.4) represents the ion and electron temperatures, normalized to the initial temperature (T_0). x and y are the directions parallel and perpendicular to the flow of the plasma. The temperature profiles also illustrate the existence of the sheath-transition region, while clearly demonstrating the role of collisionality in the plasma. In highly collisional plasmas, the charged particles collide with each other frequently, driving the x and y temperatures to be the same. This can be seen in figure (3.4a) for $K_n = 50$, which is the highly collisional case. The high collision frequency of the electrons causes the x and y electron temperatures to isotropize and become approximately equal away from the wall. Similarly, the x and y ion temperatures try to isotropize but the ions do not collide frequently enough to remove the temperature anisotropy.

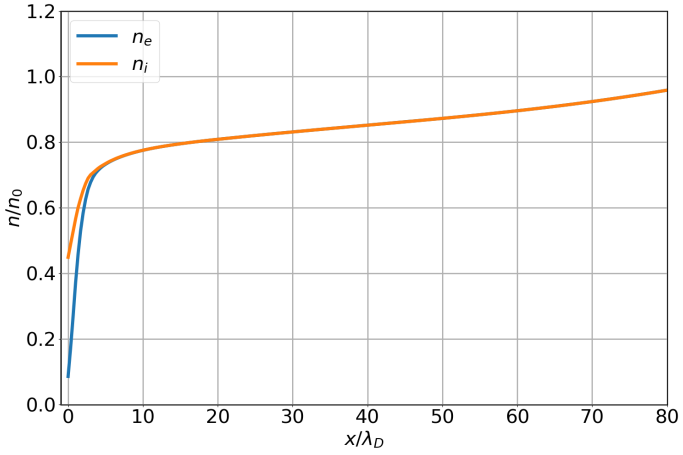
For the intermediate collisional case of $K_n = 500$, the ion temperature is even more anisotropic than the $K_n = 50$ case but there is still a very small increase in the x temperature due to some degree to isotropization. This effect is nearly negligible in the $K_n = 2000$ case, where the ion x and y temperatures seem to be unchanged throughout the domain. It can also be noted that across all collisionalities, the electrons exhibit temperature anisotropy near the wall, indicating that the fluid assumption of isothermal electrons may not be valid. Therefore, by looking at the above profiles, it is apparent that the sheath-transition region exists. This region connects the sheath to the bulk plasma and quasi-neutrality is weakly violated here.



(a) Density profile for $K_n = 50$

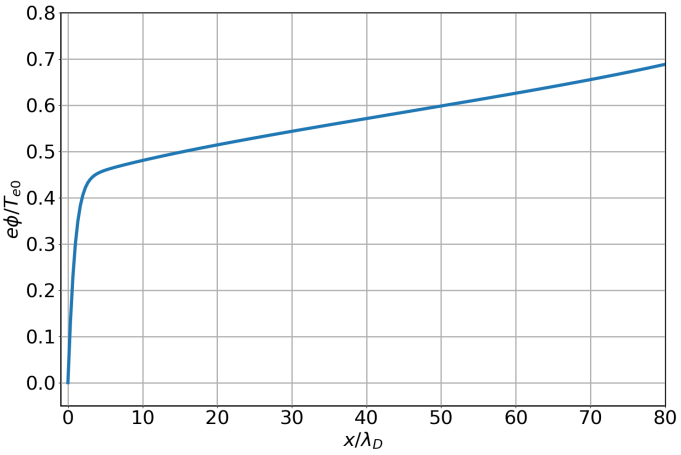


(b) Density profile for $K_n = 500$

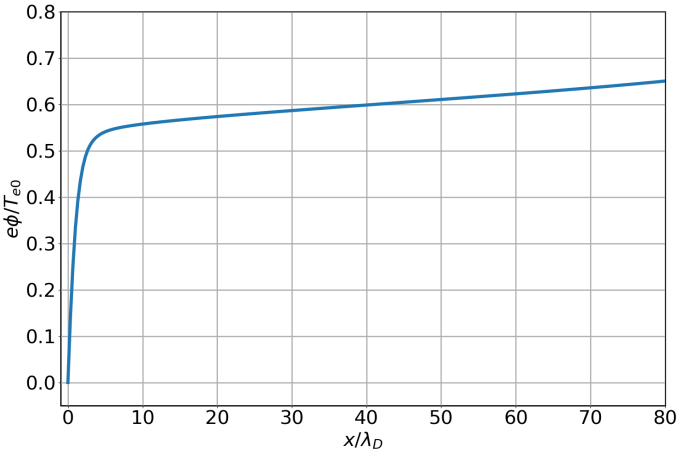


(c) Density profile for $K_n = 2000$

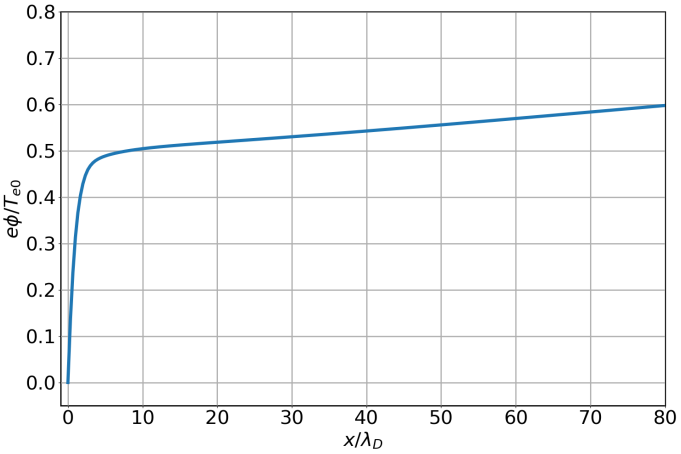
Figure 3.1: Ion and electron density profiles for Knudsen number $K_n = 50$ to $K_n = 2000$



(a) Potential profile for $K_n = 50$

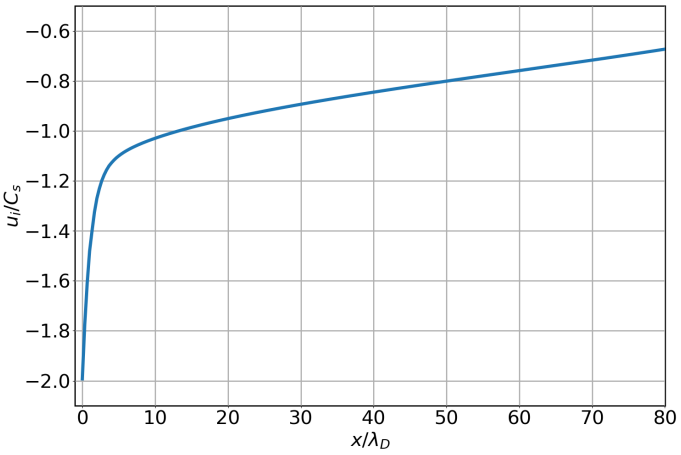


(b) Potential profile for $K_n = 500$

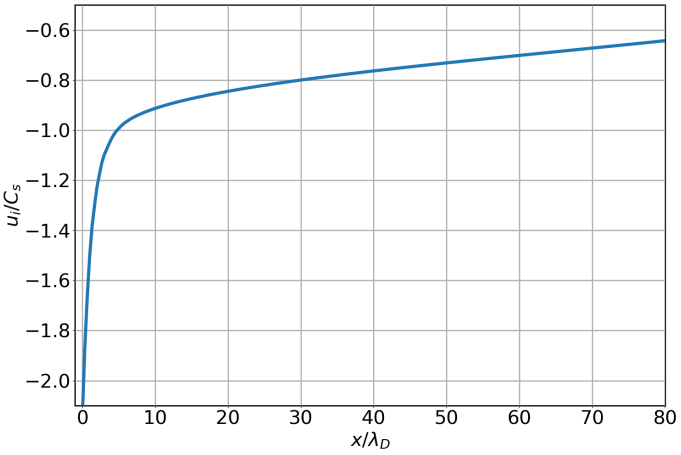


(c) Potential profile for $K_n = 2000$

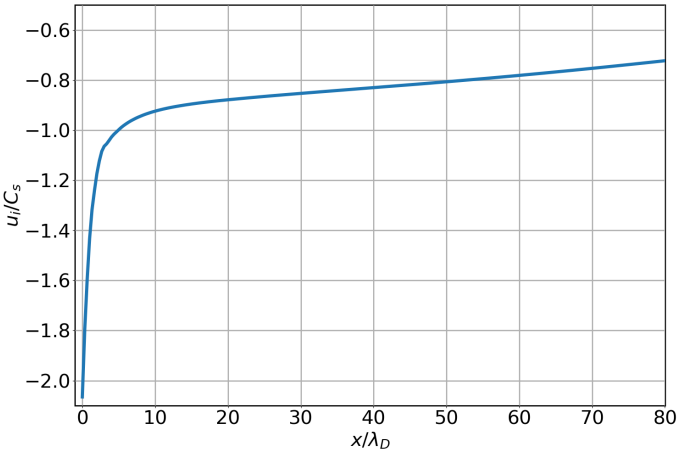
Figure 3.2: Plasma potential profiles for Knudsen number $K_n = 50$ to $K_n = 2000$



(a) Ion bulk speed profile for $K_n = 50$

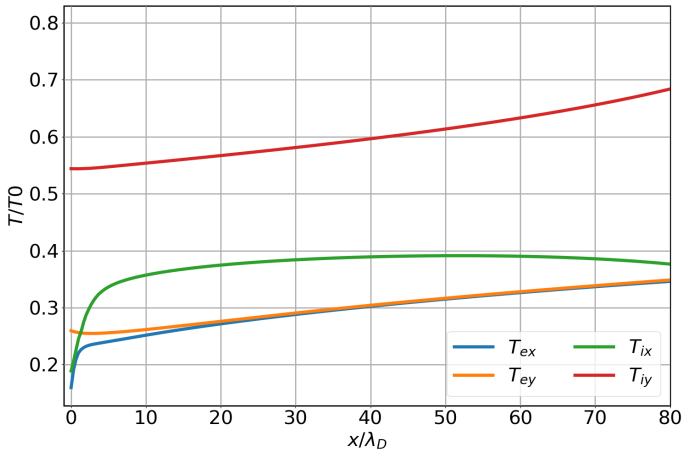


(b) Ion bulk speed profile for $K_n = 500$

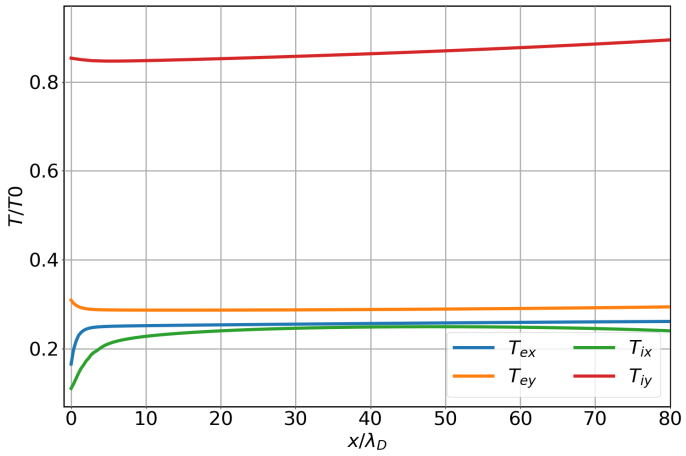


(c) Ion bulk speed profile for $K_n = 2000$

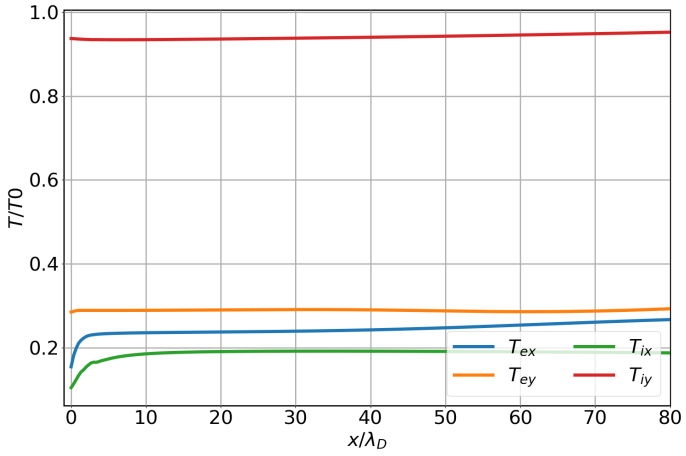
Figure 3.3: Ion bulk speed profiles for Knudsen number $K_n = 50$ to $K_n = 2000$



(a) Ion and electron temperature profile for $K_n = 50$



(b) Ion and electron temperature profile for $K_n = 500$



(c) Ion and electron temperature profile for $K_n = 2000$

Figure 3.4: Ion and electron temperature profiles for Knudsen number $K_n = 50$ to $K_n = 2000$

3.1.2 Locating the Sheath Entrance in the Sheath-transition Region

Now that the existence of the sheath-transition region has been verified, the next step in evaluating the modified Bohm speed is to find the exact location of the sheath entrance. While it is apparent that the sheath entrance must lie at the edge of the sheath-transition region, a quantitative method must be used to determine the point at which the transition region ends and the sheath begins. To locate this point, Baalrud et al. [15] suggest the use of the 'charge separation level' which is defined by,

$$\bar{\rho} = \left| \frac{n_i - n_e}{n_e + n_i} \right|. \tag{3.1}$$

They suggest that the point at which this level is evaluated to be below some subjective threshold, is the sheath entrance. This condition makes intuitive sense as it is based on the condition of the violation of quasi-neutrality. However, Li et al. [1] argue that this condition is not enough as a low charge separation level does not ensure that the Bohm criterion is satisfied. They introduce the 'fractional charge density gradient', defined as,

$$\frac{\partial \bar{\rho}}{\partial x} = \left| \frac{\partial n_e}{\partial x} - \frac{\partial n_i}{\partial x} \right| / \left| \frac{\partial n_e}{\partial x} + \frac{\partial n_i}{\partial x} \right| \tag{3.2}$$

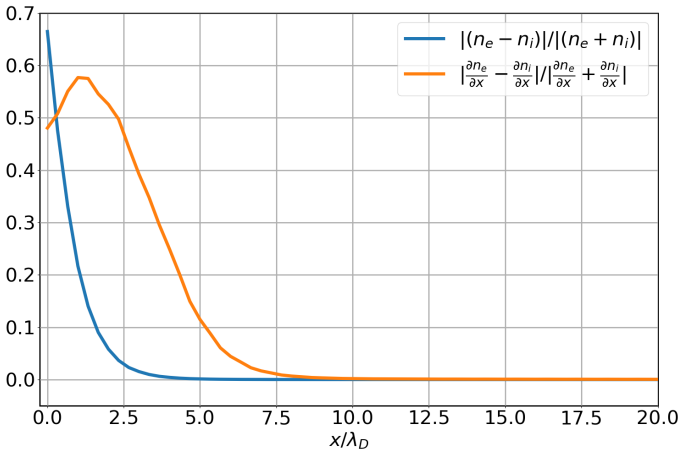
and state that for the Bohm criterion to be satisfied as in equation (2.15), this gradient must also be lower than some subjective threshold. Figure (3.5) plots both these quantities over the domain for $K_n = 50$, $K_n = 500$, and $K_n = 2000$.

Considering a distance of $2.5\lambda_D$ from the wall, for the cases of $K_n = 50$, $K_n = 50$ and $K_n = 2000$, the separation levels are well below ten percent. However, the charge gradient

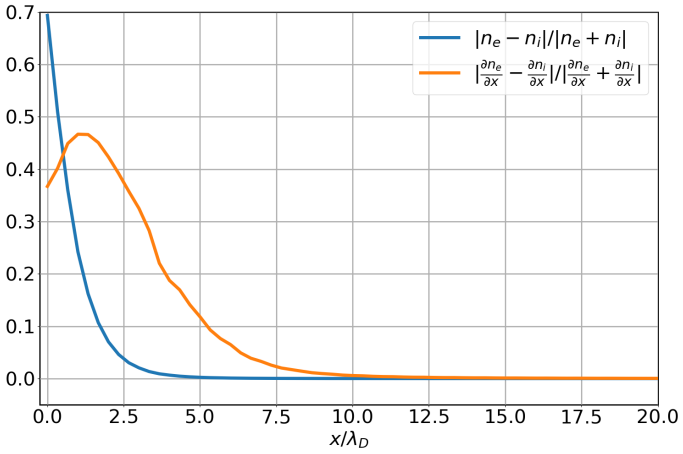
is approximately around fifty percent, thirty five percent and thirty percent for these cases respectively. Therefore, we must choose a point where both the quantities are small enough so that the sheath criterion is applicable but are also big enough to violate quasi-neutrality. Li et al. [1] chooses a threshold for ten percent for the fractional charge density gradient, at which point the sheath entrance is found to be at around $5\lambda_D$ from the wall for the three cases. The sheath entrance and normalized quantities derived from the continuum kinetic simulations for various cases is provided in table (3.1), from which we can note that the sheath entrance in roughly at $5\lambda_D$.

K_n	K_n^{se}	$\frac{T_{ey}^{se}}{T_{ix}^{se}}$	$\frac{T_{iy}^{se}}{T_{ix}^{se}}$	$\frac{eE^{se}\lambda_D}{T_{ex}^{se}}$	$\frac{x_{se}}{\lambda_D}$
20	4.728	1.031	1.277	-0.034	5.001
50	13.024	1.062	1.617	-0.021	5.334
200	58.164	1.084	2.885	-0.021	5.334
500	142.085	1.146	3.976	-0.023	5.334
1000	285.535	1.270	4.841	-0.032	4.667
2000	484.141	1.236	5.406	-0.023	5.001

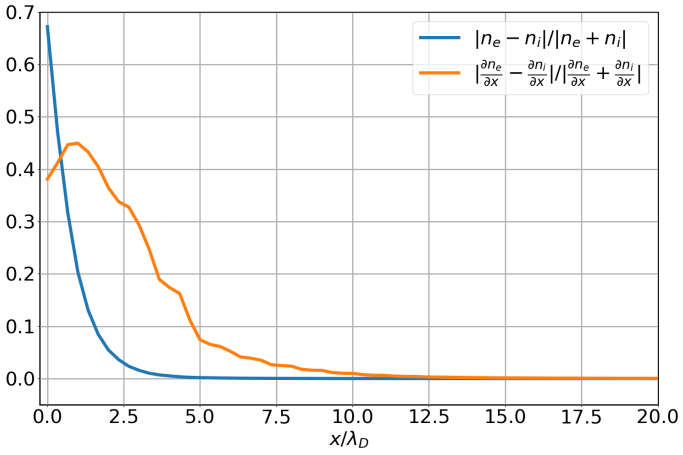
Table 3.1: Normalized parameters at the sheath entrance



(a) $\bar{\rho}$ and $\frac{\partial \bar{\rho}}{\partial x}$ for $K_n = 50$



(b) $\bar{\rho}$ and $\frac{\partial \bar{\rho}}{\partial x}$ for $K_n = 500$



(c) $\bar{\rho}$ and $\frac{\partial \bar{\rho}}{\partial x}$ for $K_n = 2000$

Figure 3.5: Charge separation level and fractional charge density gradient for $K_n = 50$ to $K_n = 2000$

3.1.3 Comparison with VPIC results

The results presented in subsections (3.1.1) and (3.1.2) have been obtained using the continuum kinetic simulation results. While the study of sheaths using this method has been performed before [33], comparing the continuum kinetic results with Li et al. [1]'s VPIC (vector particle-in-cell) results evaluates the accuracy of the continuum kinetic method. All VPIC figures and data are taken from Li et al. [1] with the permission of the corresponding author.

Figures 3.10, 3.7, 3.8 and 3.9 plot the sheath profiles using VPIC and continuum kinetics for the case of $K_n = 50$. For the density and ion speed plots, the VPIC and continuum kinetic plots are in excellent agreement as they display a similar trend over the domain. The potential developed in the continuum kinetic case is approximately 20% lower than the potential in the VPIC case. For the temperature plot, it can be noticed that the ion x, and electron x and y temperatures are in good agreement with the VPIC plot. However, the ion y temperature is much higher than the corresponding VPIC value.

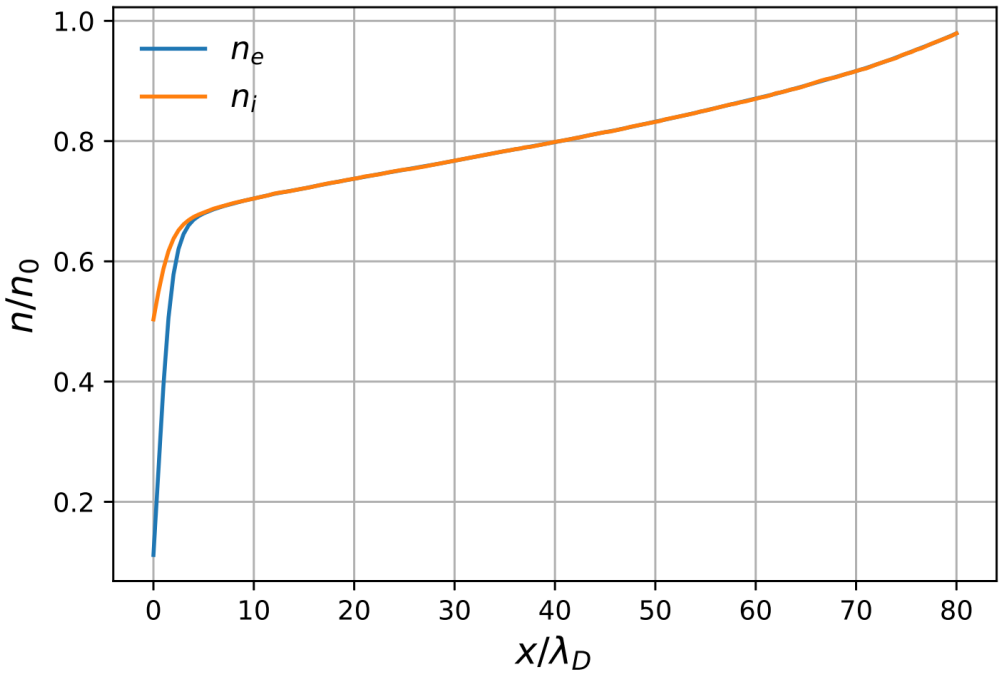
The disparity in the potentials and ion y temperatures could be due to the presence of a background magnetic field in the VPIC case and/or the difference in the collisional model between the simulations. The continuum kinetic simulations resolve the collisions using the Lenard-Bernstein collisional model (as discussed in section 2.4), which is a reduced nonlinear Fokker-Planck collision operator and the VPIC simulations uses the Takizuka-Abe collisional model [34], which captures the full nonlinear Fokker-Planck collisional model.

Figure 3.10 compares the charge separation level and the fractional charge density gradient between VPIC and the continuum kinetic method for the case of $K_n = 50$. We can see that the continuum kinetic result is in excellent agreement with the VPIC result and is free of the statistical noise present in VPIC.

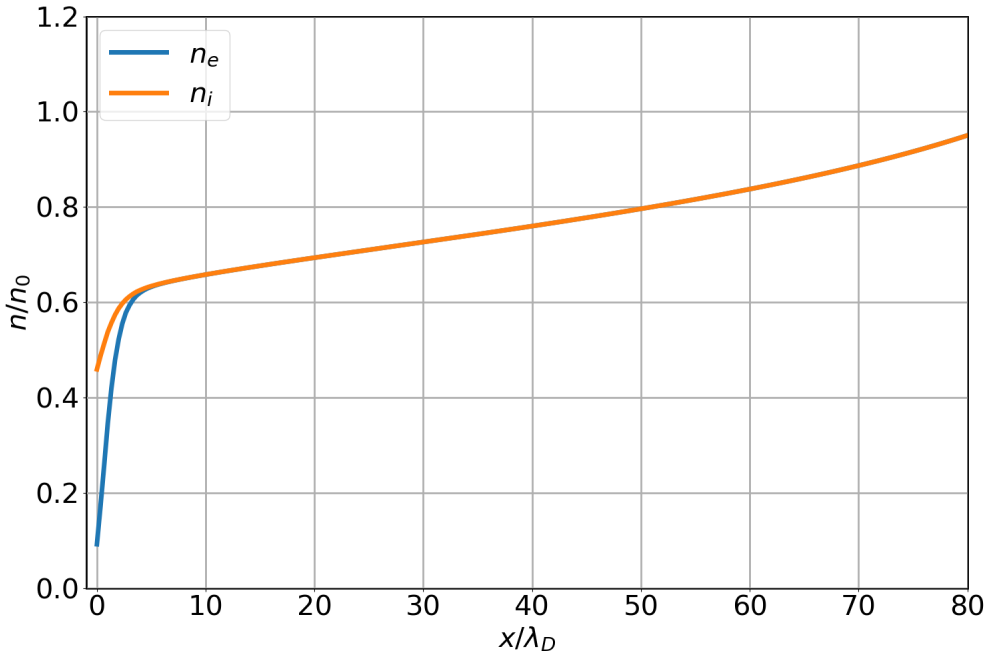
The normalized sheath profile parameters and the sheath entrance (such as in table 3.1) have been compared in table 3.2. The VPIC values are highlighted in bold text. We can see that the normalized continuum kinetic values are within reasonable agreement of the VPIC values.

K_n	K_n^{se}		$\frac{T_{ey}^{se}}{T_{ix}^{se}}$		$\frac{T_{iy}^{se}}{T_{ix}^{se}}$		$\frac{eE^{se}\lambda_D}{T_{ex}^{se}}$		$\frac{x_{se}}{\lambda_D}$	
20	4.728	**5.26**	1.031	**1.01**	1.277	**1.12**	-0.034	**-0.044**	5.001	**5.0**
50	13.024	**13.13**	1.062	**1.05**	1.617	**1.32**	-0.021	**-0.032**	5.334	**5.0**
200	58.164	**50.04**	1.084	**1.09**	2.885	**2.21**	-0.021	**-0.019**	5.334	**5.5**
500	142.085	**116.6**	1.146	**1.13**	3.976	**3.20**	-0.023	**-0.017**	5.334	**5.5**
1000	285.535	**220.4**	1.270	**1.19**	4.841	**4.03**	-0.032	**-0.016**	4.667	**5.5**
2000	484.141	**420.9**	1.236	**1.29**	5.406	**4.66**	-0.023	**-0.014**	5.001	**5.0**

Table 3.2: Comparison between normalized sheath profile parameters at the sheath entrance between VPIC (in bold) and continuum kinetic method. VPIC data taken from [1] with permission of corresponding author

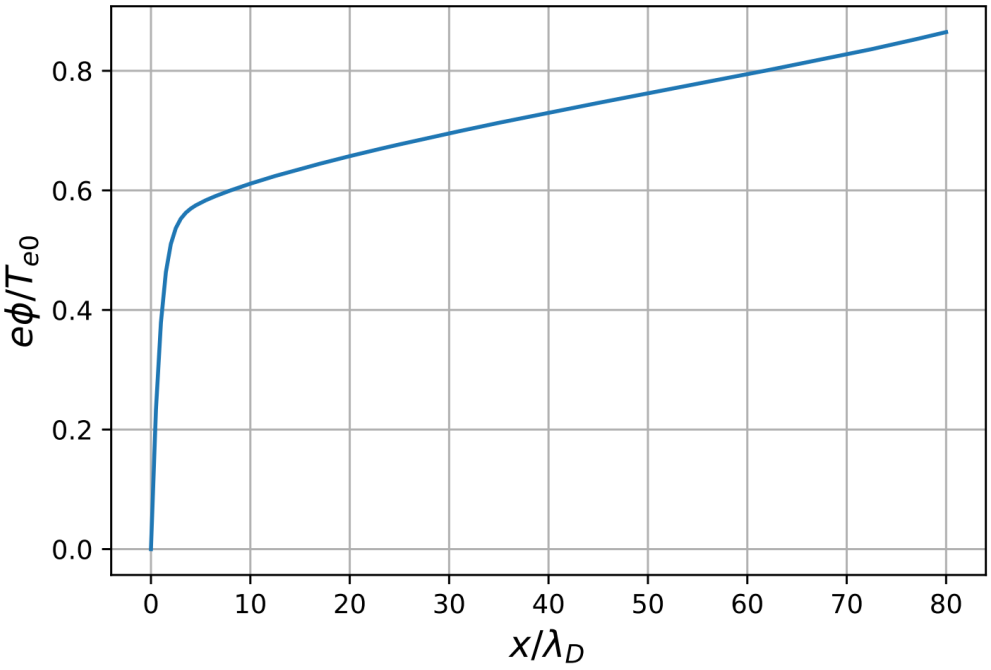


(a) VPIC density profile for $K_n = 50$. VPIC figure taken from [1] with permission of corresponding author

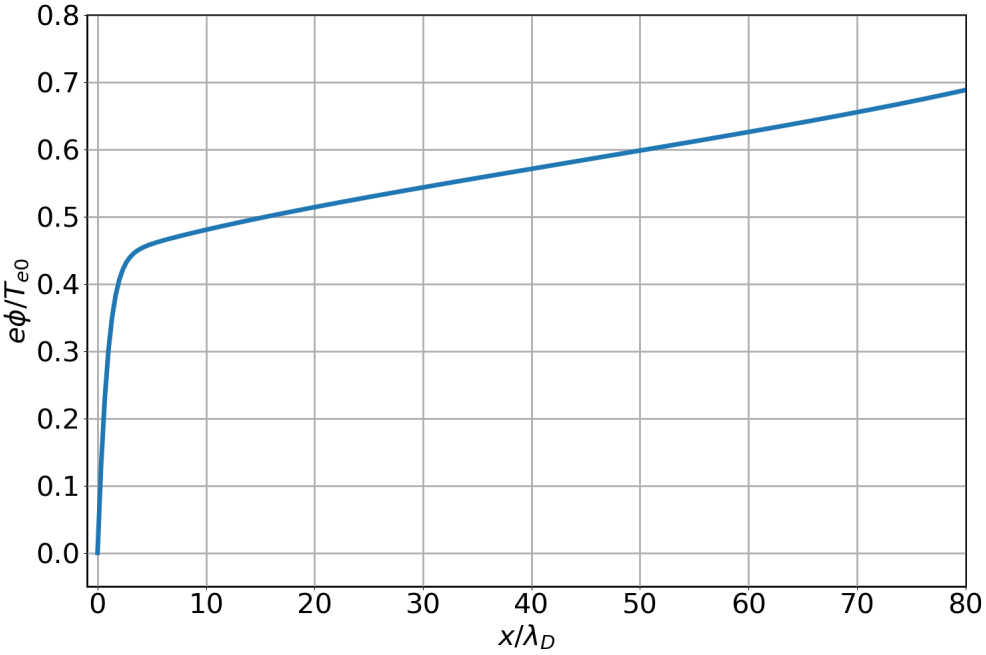


(b) Continuum kinetic density profile for $K_n = 50$

Figure 3.6: VPIC and continuum kinetic density profile for $K_n = 50$

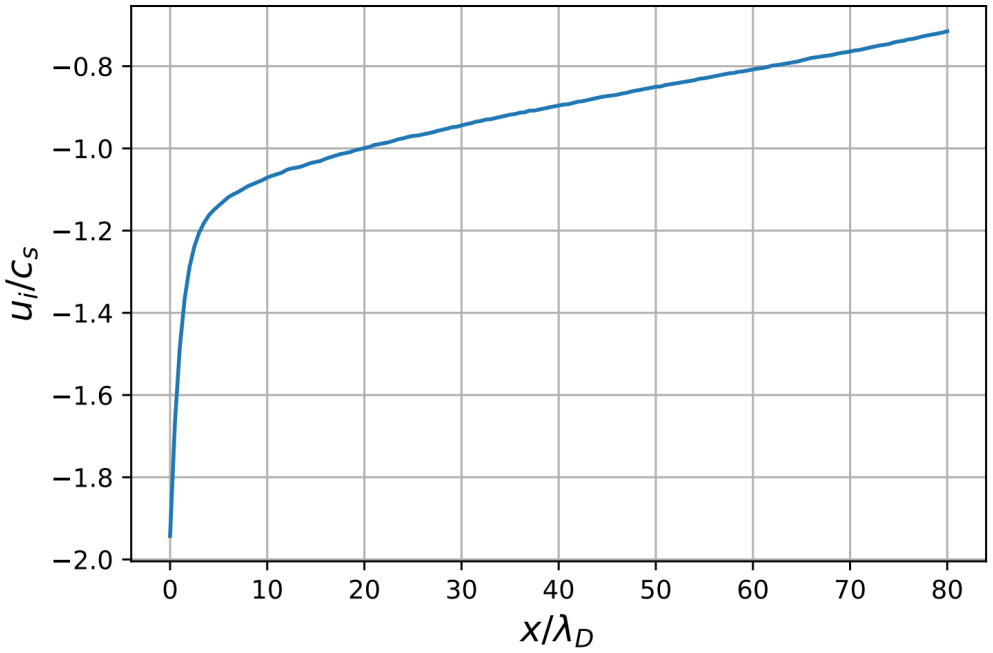


(a) VPIC potential profile for $K_n = 50$. VPIC figure taken from [1] with permission of corresponding author

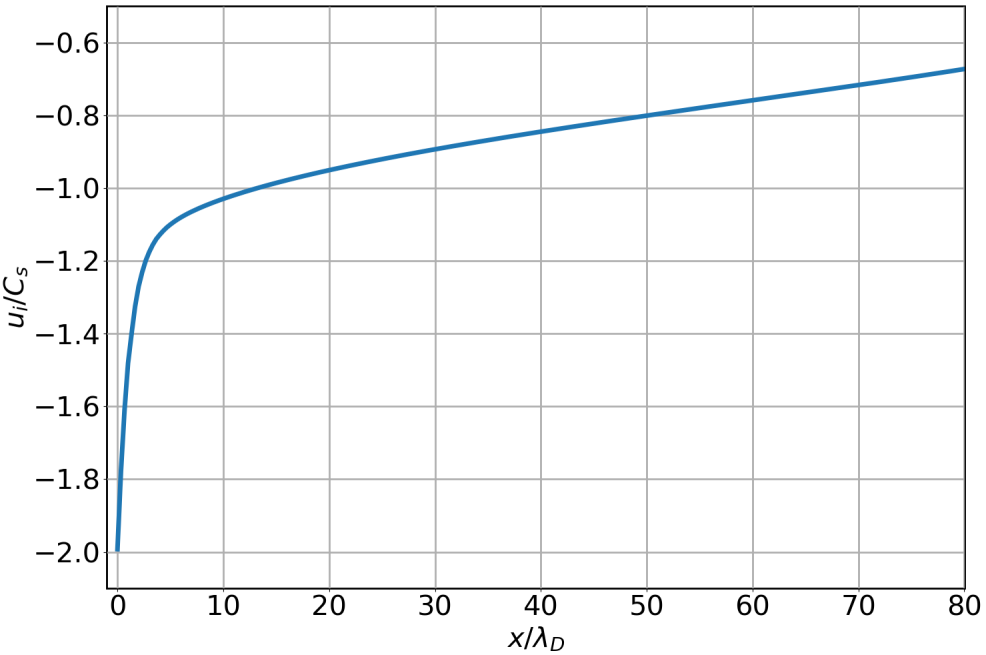


(b) Continuum kinetic potential profile for $K_n = 50$

Figure 3.7: VPIC and continuum kinetic potential profile for $K_n = 50$

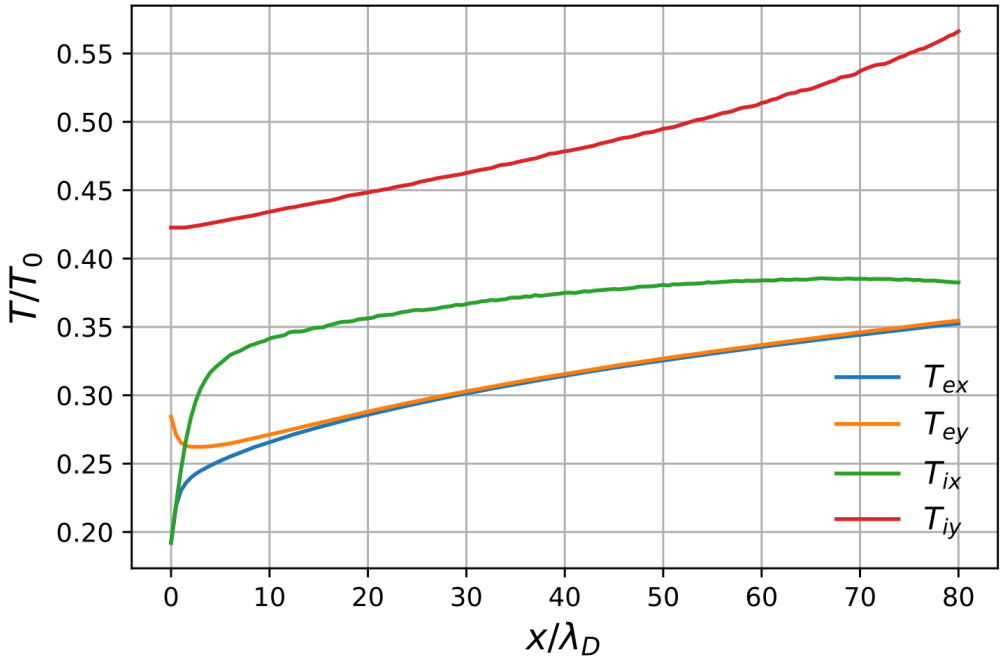


(a) VPIC ion bulk speed profile for $K_n = 50$. VPIC figure taken from [1] with permission of corresponding author

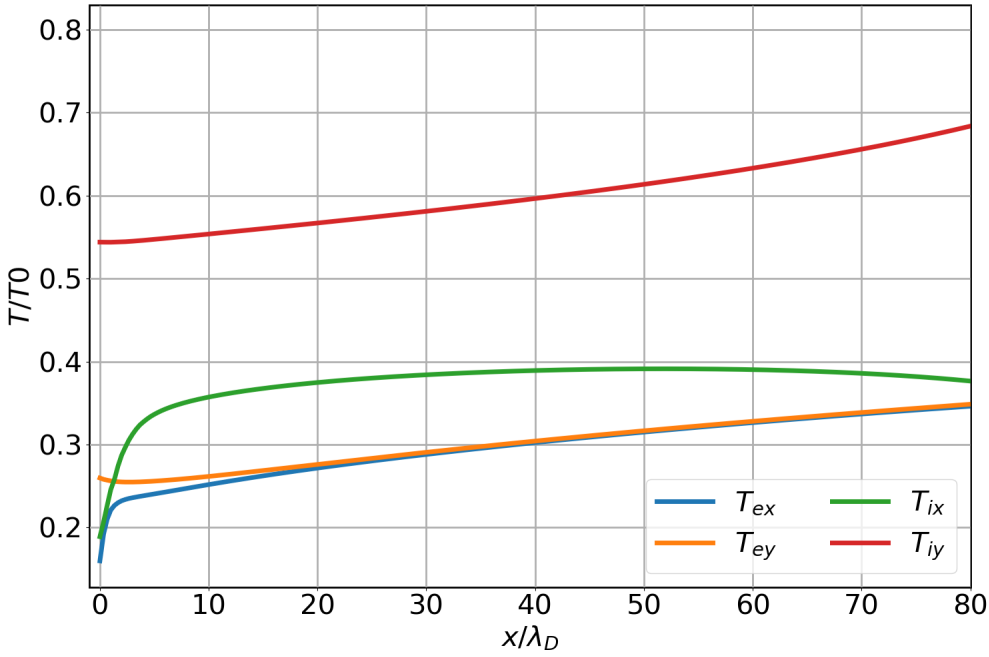


(b) Continuum kinetic ion bulk speed profile for $K_n = 50$

Figure 3.8: VPIC and continuum kinetic ion speed profile for $K_n = 50$

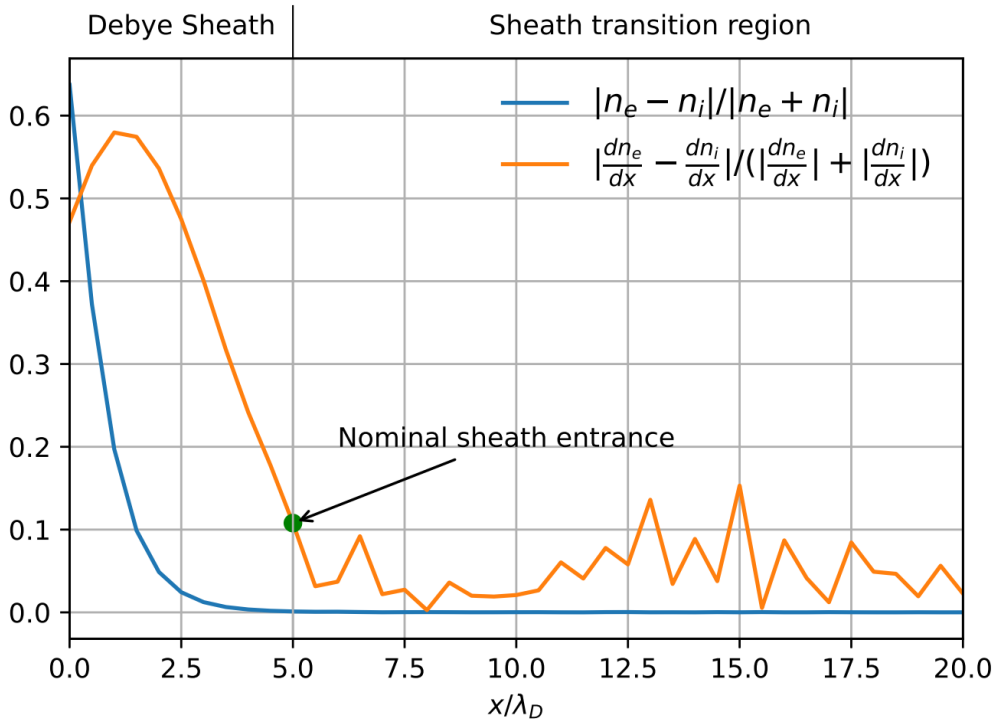


(a) VPIC temperature profile for $K_n = 50$. VPIC figure taken from [1] with permission of corresponding author

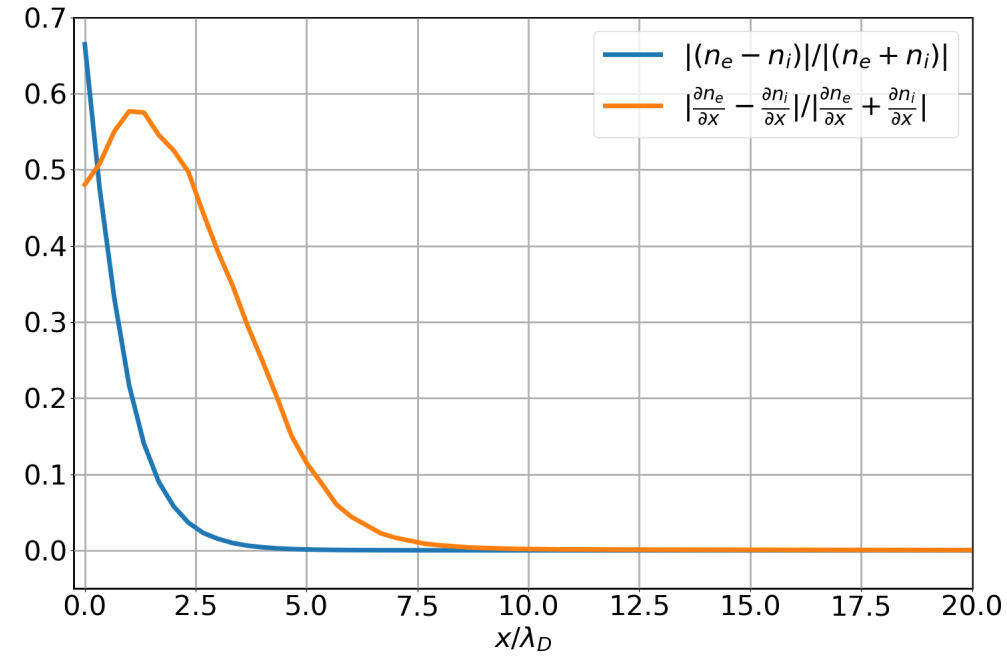


(b) Continuum kinetic temperature profile for $K_n = 50$

Figure 3.9: VPIC and continuum kinetic temperature profile for $K_n = 50$



(a) VPIC $\bar{\rho}$ and $\frac{\partial \bar{\rho}}{\partial x}$ for $K_n = 50$. VPIC figure taken from [1] with permission of corresponding author



(b) Continuum kinetic $\bar{\rho}$ and $\frac{\partial \bar{\rho}}{\partial x}$ for $K_n = 50$

Figure 3.10: VPIC and continuum kinetic charge separation level and fractional charge density gradient for $K_n = 50$

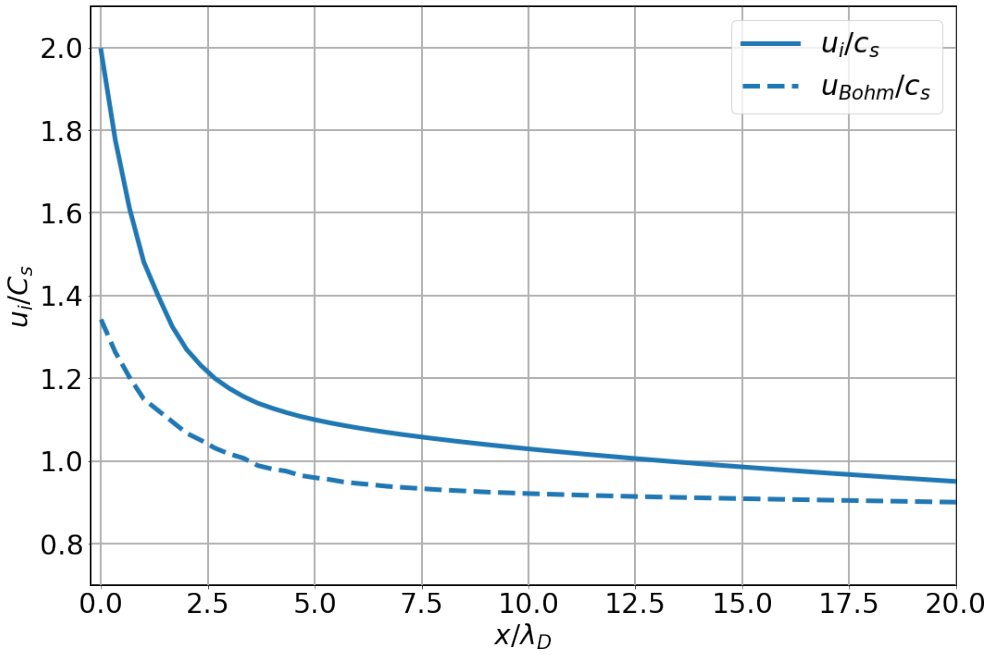
3.2 The Modified Bohm speed

With a quantitative way to find the sheath entrance in cases where the sheath-transition region exists, the transport terms in equation 2.18 can now be evaluated. Using the simulation outputs in the equations 2.21, 2.22, 2.20 and 2.23, the heat flux terms (q_e, q_i), the thermal force term (α), and the isotropization terms ($Q_{ei}, Q_{ee}, Q_{ie}, Q_{ii}$) are determined respectively. These terms are then substituted into equation 2.17 to find the modified Bohm speed.

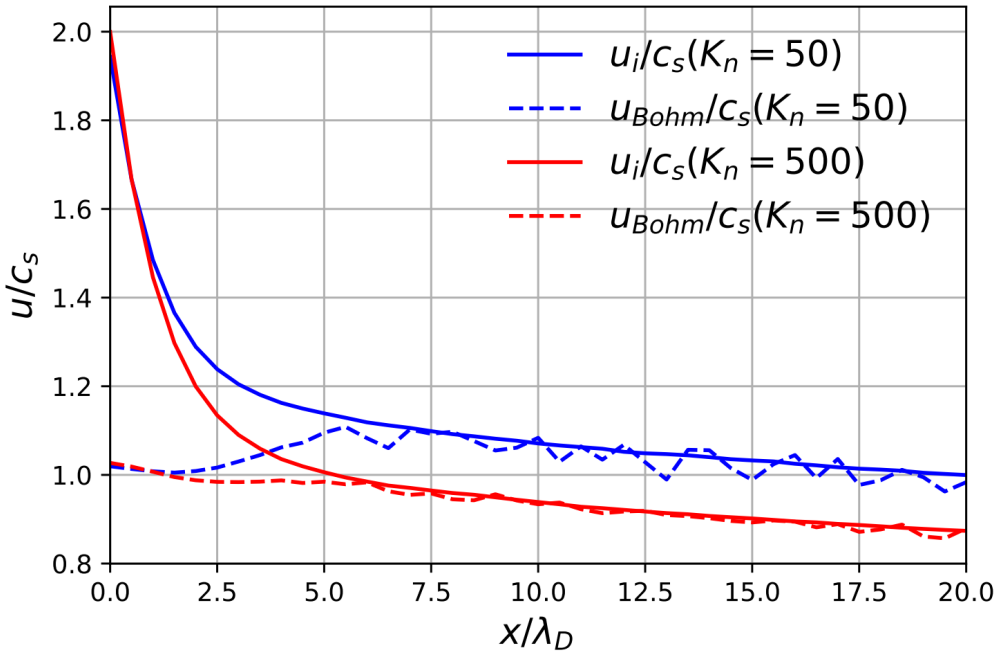
Figure 3.11a compares the ion flow speed and the modified Bohm speed for the case of $K_n = 50$ using continuum kinetics, and figure 3.11b compares the ion flow speed and the modified Bohm speed for the case of $K_n = 50$ and $K_n = 500$ using VPIC. We can see that for the case of $K_n = 50$, the modified Bohm speed model accurately predicts the ion flow speed till the sheath entrance in the VPIC simulations. This sort of agreement is not found in the continuum kinetic simulations. The overall trend of the model is similar to how the ion speed develops, but the model underestimates the ion flow velocity by approximately 5%.

In section 3.1 we have seen that the sheath profiles from the VPIC simulations and the continuum kinetic simulations were quite similar and agreed well with each other. The exceptions to that agreement were the plasma potential and the ion y temperature. Based on those profiles it was concluded that the continuum kinetic method captures much of the same physics as the PIC method. The discrepancy between the VPIC and continuum kinetic Bohm speed prediction results may be due to the difference in the way collisions are modeled in the VPIC and continuum kinetic simulations.

The collisional model is responsible for computing the change of the distribution function with time ($\frac{\partial f}{\partial t}$). This $\frac{\partial f}{\partial t}$ term plays a significant role in the time evolution of the plasma and directly affects the thermal force and the isotropization terms as can be seen from equations



(a) Comparison between ion flow speed and modified Bohm speed for $K_n = 50$ derived using continuum kinetics



(b) Comparison between ion flow speed and modified Bohm speed for $K_n = 50$ and $K_n = 500$ derived using VPIC. VPIC figure taken from [1] with permission of corresponding author

Figure 3.11: VPIC and continuum kinetic comparison of the ion flow velocity and modified Bohm speed.

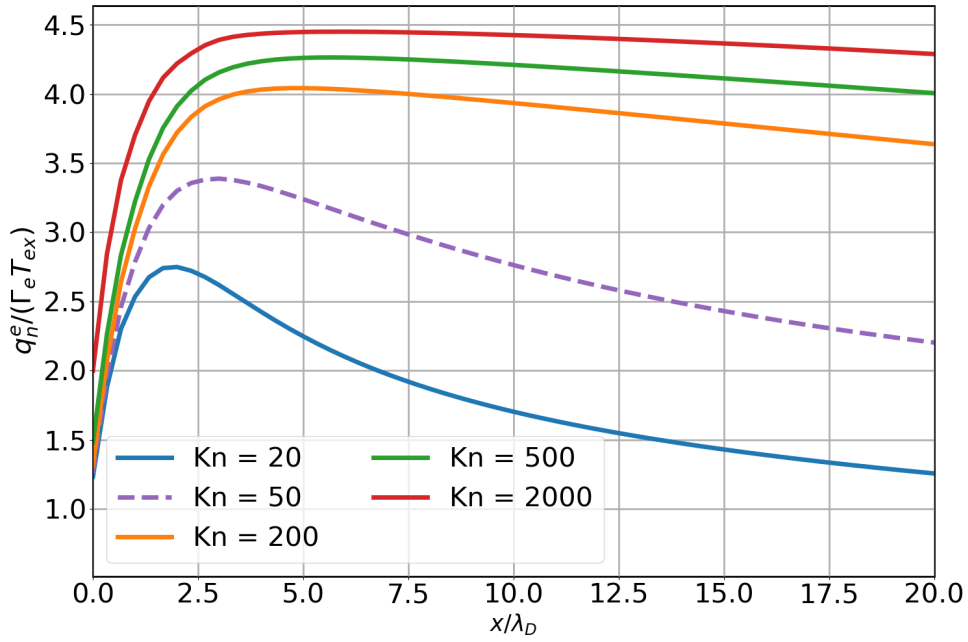
(2.19) and (2.23). Therefore, these terms and consequently the Bohm speed have a direct dependence on the collisional model being used. The thermal force and isotropization terms are also present in the momentum and energy equations and hence also affect the plasma transport. To investigate the role of the collisional model, we look at the transport terms. i.e., the heat fluxes, thermal force and isotropization terms.

Figure 3.12 compares the electron heat flux (normalized to the electron particle flux times the electron x temperature) from the VPIC and continuum kinetic simulations. The heat flux is plotted over multiple collisionalities. For the highly collisional case of $K_n = 20$. the electron heat flux across both methods are similar. As the collisionality decreases, the VPIC simulations generally report a higher electron heat flux value than what is reported by the continuum kinetic simulations. For the case of $K_n = 2000$ the VPIC normalized electron heat flux value is well above 4.5 till twenty Debye lengths, but for the same collisionality, the normalized heat flux does not exceed 4.5 in the continuum kinetic case.

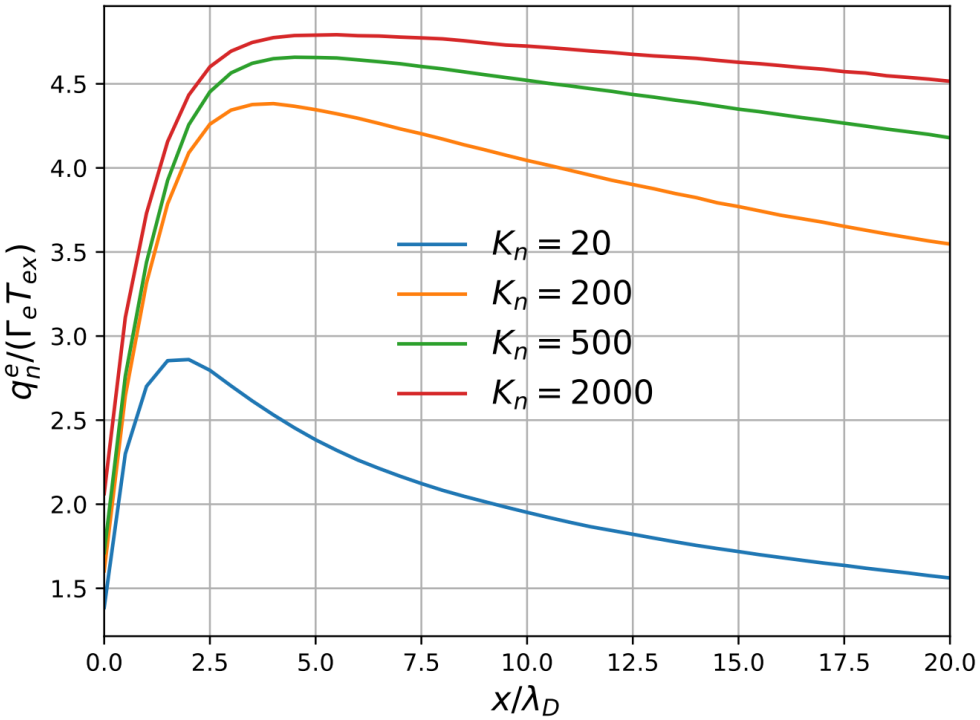
The thermal force is represented by α, the thermal force coefficient and is plotted in figure 3.13, along with the isotropization terms for $K_n = 50$. The impact of the collisional model is most pronounced in this figure, as α and the isotropization terms are directly dependant on $(\partial f/\partial t)$ according to equations (2.19) and (2.23).

In figure 3.13, the most obvious difference between the results of the two methods is the magnitude of the electron-electron isotropization term (Q_{ee}) and the electron-ion isotropization term (Q_{ei}) from the continuum kinetic simulations. The value of these terms are more than twice the value obtained from VPIC simulations and may lead to significant differences in the evaluation of the Bohm speed. The negligible thermal force in the continuum kinetics simulations can be noticed here as well, as opposed to the negative value of α in figure 3.13b. The electron heat flux, ion heat flux and ion-ion isotropization terms (Q_{ii}) are reasonably accurate.

Hence, when compared to the results of the VPIC simulation, the electron-electron isotropization term, electron-ion isotropization term and the thermal force term are the most dissimilar to their VPIC counterparts. These terms directly depend on the collisional derivative, $(\frac{\partial f}{\partial t})$ and consequently are heavily dependent on the collisional model.

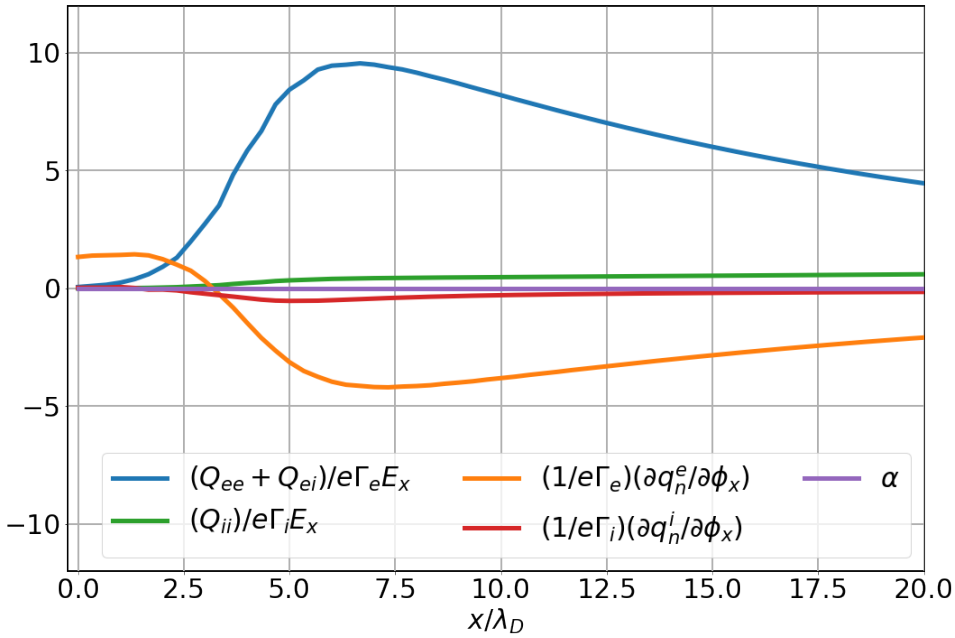


(a) Electron heat flux using continuum kinetics

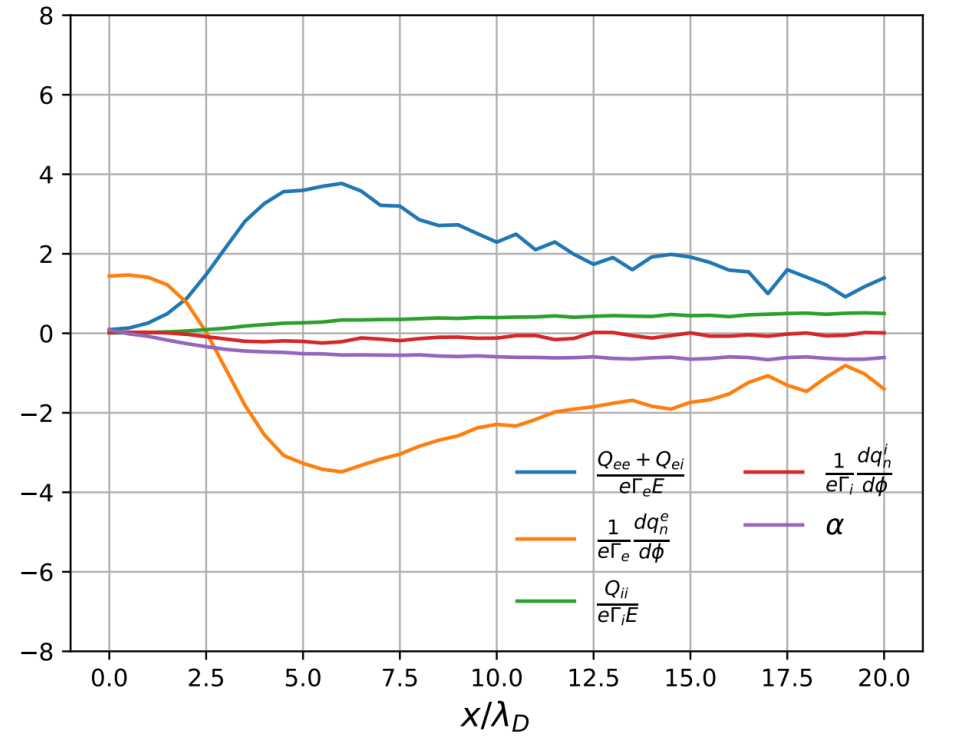


(b) Electron heat flux derived using VPIC. VPIC figure taken from [1] with permission of corresponding author

Figure 3.12: VPIC and continuum kinetic comparison of the normalized electron heat flux across multiple collisionalities.



(a) Normalized transport terms for $K_n = 50$ derived from continuum kinetics



(b) Normalized transport terms for $K_n = 50$ derived from VPIC. VPIC figure taken from [1] with permission of corresponding author

Figure 3.13: Comparison of normalized transport terms for $K_n = 50$ between VPIC and continuum kinetics

3.2.1 Comparison between Collision models

In section 2.4, it was mentioned that the Lenard-Bernstein collision model (LBO) has been used to compute the collision integral and evolve the distribution functions. As the isotropization terms and thermal force term directly depend on this collisional derivative, they have been evaluated with the same model. In the previous section, we saw that the isotropization and thermal force terms that have been evaluated with the LBO model are different to that obtained from the VPIC simulations. As these terms directly depend on the collisional derivative, we would like to know if the difference in the terms is due to the collisional model that has been used. To test this, we now consider the Bhatnagar-Gross-Krook (BGK) collisional model [35] as described in 2.4

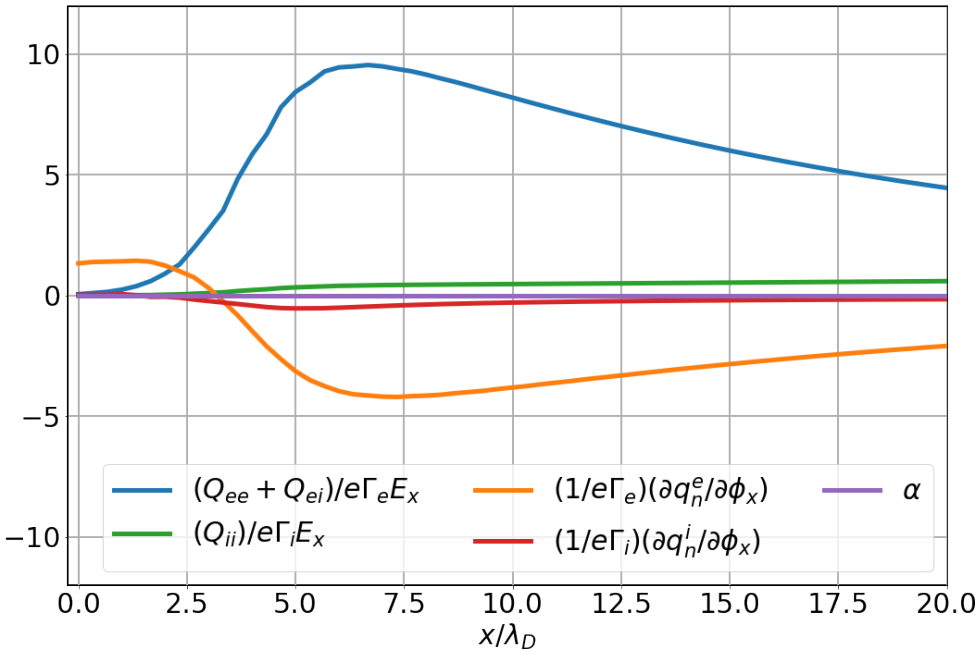
Using the BGK model we can now compute the collisional derivative, the isotropization, and the thermal force terms. Figure 3.14 compares the normalized transport terms that have been obtained by the BGK model with the LBO model for $K_n = 50$. When compared to the normalized transport terms from the VPIC simulation in figure 3.13b, the BGK derived electron-electron and electron-ion isotropization terms are closer to the VPIC solution than the LBO model. Unlike the LBO model, The BGK model produces a negligible result for the ion-ion isotropization term. Like the LBO model, however, the BGK model also computes the thermal force coefficient α to be negligible. The transport terms from the LBO and BGK model are compared with the VPIC results in table 3.3.

Figure 3.15 compares the modified Bohm speeds predictions using the LBO model and the BGK model. Although the LBO model was found to underestimate the ion flow speed, the BGK model overestimates the ion flow speed and accurately predicts the ion flow speed only at the sheath entrance. Nevertheless, we can clearly see that the modified bohm speed model is significantly affected by the collisional model. Hence it is likely that the discrepancy

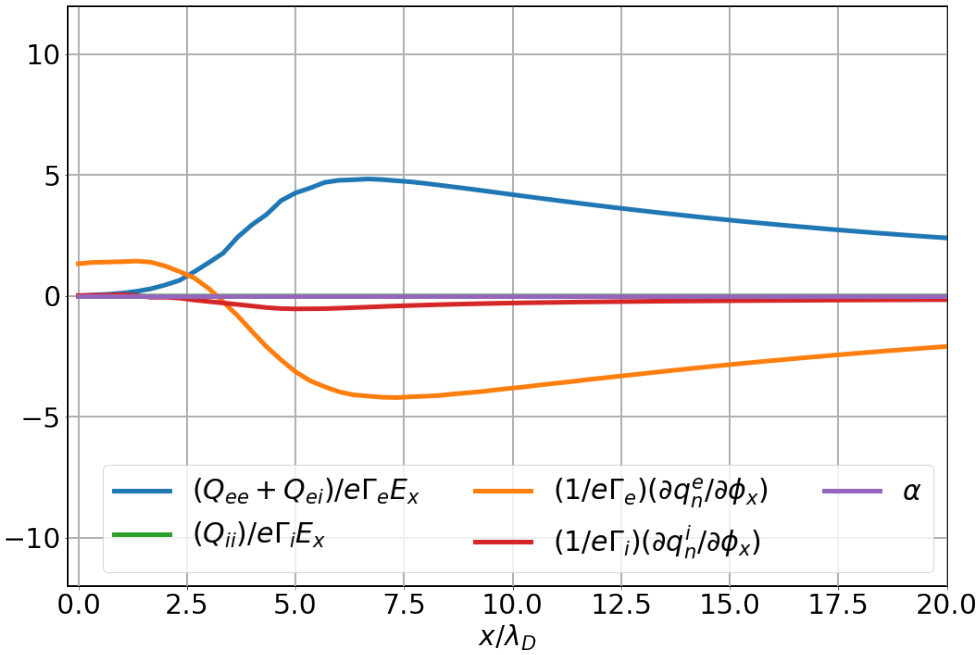
between the continuum kinetic results and the VPIC results is due to the difference of collisional models.

	$\frac{u_{ix}}{c_s}$	$\frac{u_{Bohm}}{c_s}$	$\frac{1}{e\Gamma_e^{se}}\frac{\partial q_n^e}{\partial \phi_x}$	$\frac{Q_{ee}^{se}+Q_{ei}^{se}}{e\Gamma_e^{se}E^{se}}$	$\frac{1}{e\Gamma_i^{se}}\frac{\partial q_n^i}{\partial \phi_x}$	$\frac{Q_{ii}^{se}}{e\Gamma_i^{se}E^{se}}$	α^{se}
VPIC	1.14	1.10	-3.27	3.59	-0.20	0.26	0.51
LBO	1.09	0.95	-3.50	8.83	-0.52	0.36	-5.04×10^{-10}
BGK	1.09	1.11	-3.50	4.46	-0.52	1.05×10^{-4}	-1.44×10^{-10}

Table 3.3: Comparison of normalized transport terms derived from the LBO and BGK collisional model to VPIC results for $K_n = 50$. VPIC data taken from [1] with permission of corresponding author

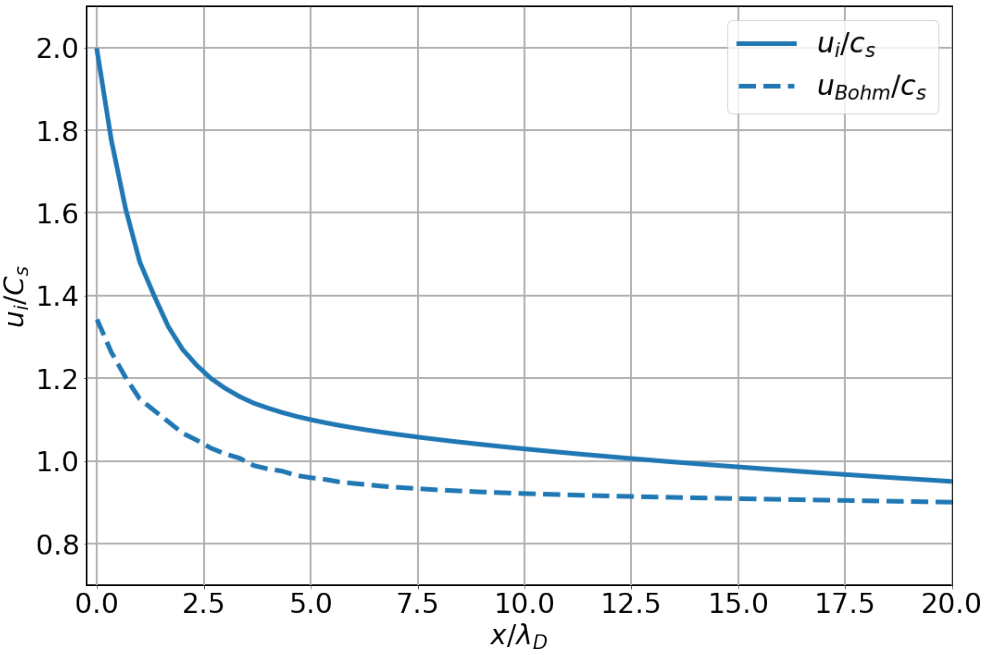


(a) Normalized transport terms for $K_n = 50$ using the LBO model

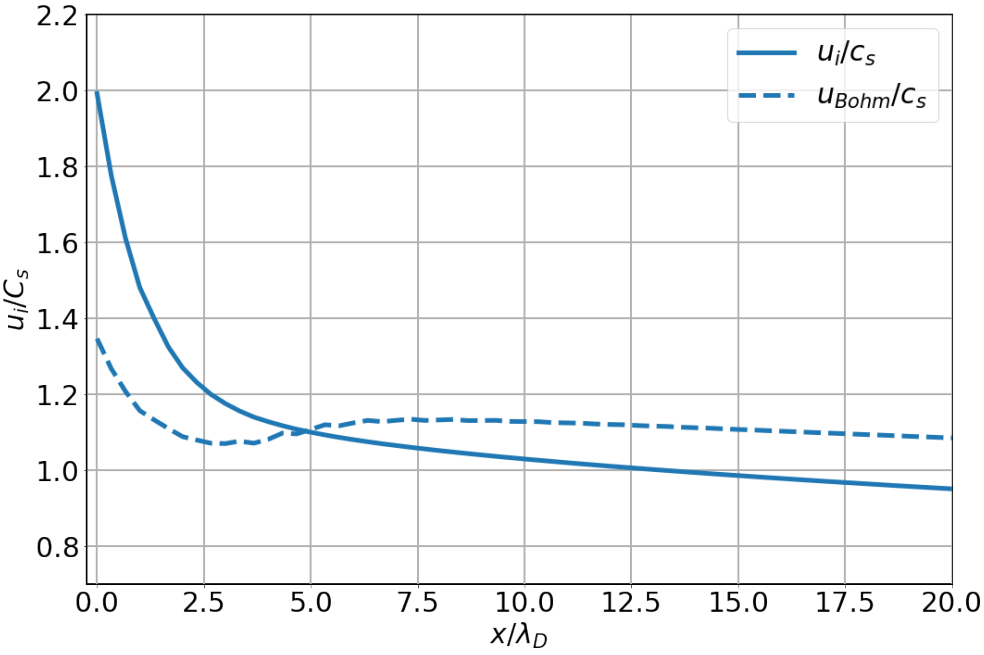


(b) Normalized transport terms for $K_n = 50$ using the BGK model

Figure 3.14: Comparison of normalized transport terms for $K_n = 50$ between the LBO and BGK collisional models



(a) Comparison between ion flow speed and modified Bohm speed using the LBO model for $K_n = 50$



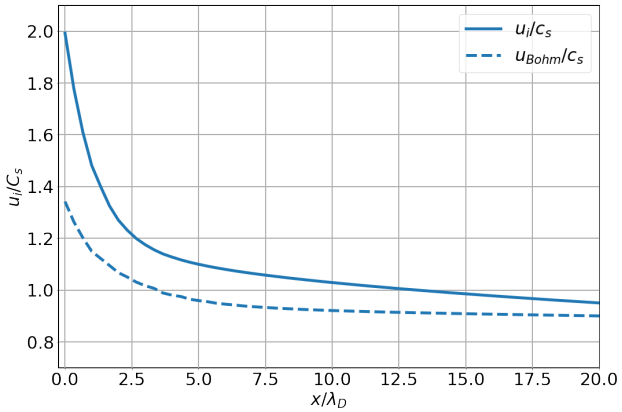
(b) Comparison between ion flow speed and modified Bohm speed using the BGK model for $K_n = 50$

Figure 3.15: Comparison of modified Bohm speed between the LBO and BGK models

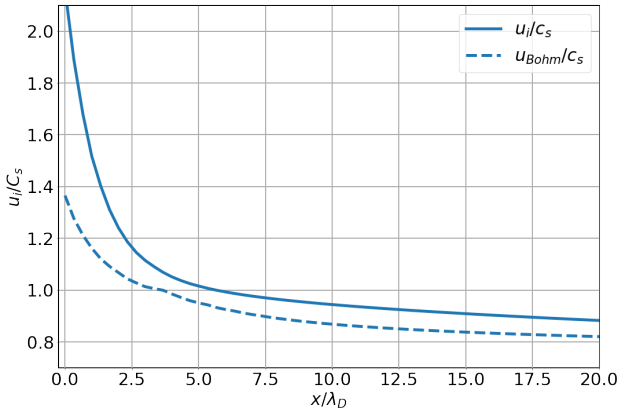
3.2.2 Effect of Collisionality on the Modified Bohm Speed

In the previous section, we saw that the collisional model used to compute the collisional derivative significantly impacts the modified Bohm speed prediction. The comparison between the LBO and BGK models was conducted for the case of $K_n = 50$, which is highly collisional. We would now like to investigate how the prediction changes with collisionality. Figure 3.16 plots the ion flow speed and the modified Bohm speed for the cases of $K_n = 50$, $K_n = 500$, and $K_n = 2000$ using the LBO model. Similarly, 3.17 plots the ion flow speed and the modified bohm speed for the cases of $K_n = 50$, $K_n = 500$, and $K_n = 2000$ using the BGK model.

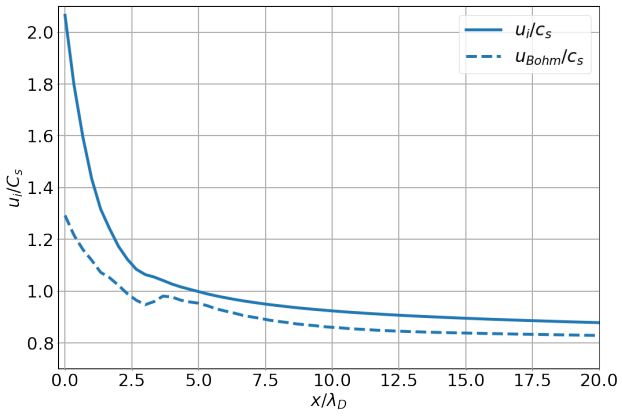
For the LBO as well as the BGK collisional models, the Bohm speed predictions seem to get more accurate with decreasing collisionality. The LBO and BGK model continue to underestimate and overestimate the ion flow speed respectively. Table 3.4 lists the normalized transport terms as well as the ion speed and the Bohm speed at the sheath entrance for the LBO model.



(a) Ion flow speed and modified Bohm speed for $K_n = 50$ using the LBO model

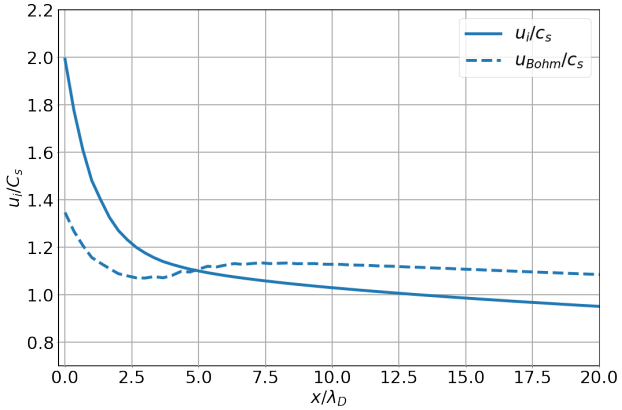


(b) Ion flow speed and modified Bohm speed for $K_n = 500$ using the LBO model

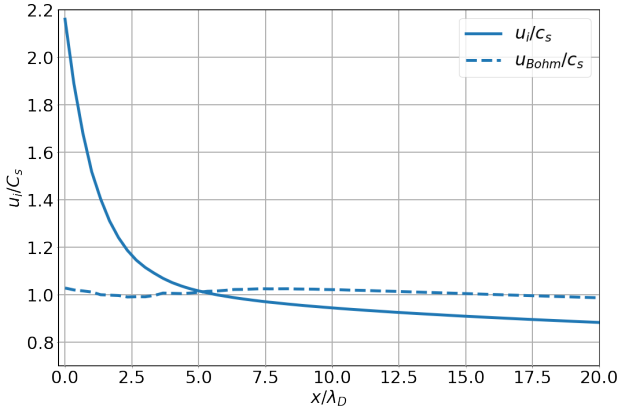


(c) Ion flow speed and modified Bohm speed for $K_n = 2000$ using the LBO model

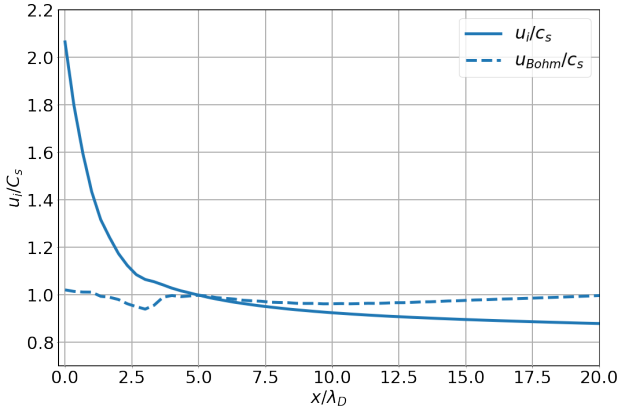
Figure 3.16: Variation of Ion flow speed and Bohm speed predictions with collisionality in the domain of $K_n = 50$ to $K_n = 2000$ using the BGK model



(a) Ion flow speed and modified Bohm speed for $K_n = 50$ using the BGK model



(b) Ion flow speed and modified Bohm speed for $K_n = 500$ using the BGK model



(c) Ion flow speed and modified Bohm speed for $K_n = 2000$ using the BGK model

Figure 3.17: Variation of Ion flow speed and Bohm speed predictions with collisionality in the domain of $K_n = 50$ to $K_n = 2000$ using the BGK model

K_n	$\frac{u_{ix}}{c_s}$	$\frac{u_{Bohm}}{c_s}$	$\frac{1}{e\Gamma_e^{se}}\frac{\partial q_n^e}{\partial \phi_x}$	$\frac{Q_{ee}^{se}+Q_{ei}^{se}}{e\Gamma_e^{se}E^{se}}$	$\frac{1}{e\Gamma_i^{se}}\frac{\partial q_n^i}{\partial \phi_x}$	$\frac{Q_{ii}^{se}}{e\Gamma_i^{se}E^{se}}$	α^{se}
20	1.22	0.97	-3.17	6.79	-0.24	0.17	-5.05×10^{-10}
	1.25	**1.20**	**-1.63**	**1.48**	**-0.05**	**0.18**	**0.59**
50	1.09	0.95	-3.50	8.83	-0.52	0.36	-5.04×10^{-10}
	1.14	**1.10**	**-3.27**	**3.59**	**-0.20**	**0.26**	**0.51**
200	1.00	0.95	-0.02	3.31	-0.20	0.26	-2.64×10^{-10}
	1.02	**1.00**	**-2.20**	**3.55**	**-0.22**	**0.38**	**0.45**
500	1.00	0.94	0.32	2.65	0.02	0.16	-1.53×10^{-10}
	0.99	**0.98**	**-0.68**	**2.29**	**-0.08**	**0.32**	**0.39**
1000	0.99	0.94	0.89	1.56	0.16	0.06	-3.59×10^{-11}
	0.98	**0.96**	**-0.18**	**1.69**	**0.14**	**0.23**	**0.18**
2000	0.99	0.95	0.90	1.41	0.15	0.05	-5.15×10^{-10}
	0.95	**0.94**	**0.44**	**1.36**	**0.36**	**0.12**	**0.11**

Table 3.4: Normalized transport terms for $Kn = 20$ to $K_n = 2000$ using the LBO model, compared to the VPIC results (in bold). VPIC figure taken from [1] with permission of corresponding author

Chapter 4

Conclusions and Future Work

The Bohm sheath criterion and the Bohm speed establish the necessary conditions for the formation of the plasma sheath. Classical sheath theory provides the analytical formulation of the Bohm speed, which is expressed using the heat capacity ratios. This expression however, is only accurate in the limit of $\lambda_D/L \to 0$. Away from this limit, matched two-scale analysis shows the existence of a sheath-transition layer. In the cases where this transition layer is present, the modified Bohm speed formulation is derived by considering the role of transport physics and incorporating transport terms such as the thermal force, particle heat fluxes, and temperature isotropization terms. Kinetic simulations have been performed by solving the Vlasov-Maxwell system of equations using the discontinuous Galerkin method in the Gkeyll software framework. This method is called the continuum kinetic approach. The existence of the sheath transition region is verified and the location of the sheath entrance is found in this region. The continuum kinetic approach is benchmarked against the VPIC results and is found to satisfactorily capture much of the physics, such as the density, electron temperature ion speed while offering noise-free solutions. There are some differences however, in the ion y temperature and the plasma potential. The modified Bohm speed is evaluated using the continuum kinetic results and is found to be inaccurate. The reason behind the disparity in the continuum kinetic method and VPIC method is investigated and a difference in the isotropization and thermal force terms are found. These terms are calculated from the collisional integral, and consequently the collisional model. It is hypothesized that the

disparity arises due to the difference in the nonlinear Takizuka-Abe collisional model used in the VPIC simulation and the reduced Lenard-Bernstein collisional model used in the continuum kinetic simulations.

4.1 Future Work

While the continuum kinetic results are within reasonable agreement to the VPIC results and capture the physics well, there are still a few discrepancies in the plasma profile. Specifically, the potential developed in the continuum kinetic case is notably lower than that found in the VPIC results. A possible reason for this could be the exclusion of the background magnetic field. The background magnetic field was included in the VPIC simulations to suppress the Weibel instability, but since the instability does not form in the time scales of the continuum kinetic simulations, the field has been neglected. Conducting a continuum kinetic study with the presence of the background magnetic field may be beneficial in understanding the role of this field.

More importantly, the effects of the collisional model must be studied further due to the disagreement in the continuum kinetic and VPIC results. Implementing a full nonlinear Fokker-Planck collisional operator may produce results that agree more with the VPIC results, as it would provide a direct comparison to the nonlinear Takizuka-Abe collisional operator used in the VPIC simulations.

Bibliography

[1] Yuzhi Li, Bhuvana Srinivasan, Yanzeng Zhang, and Xian-Zhu Tang. Transport physics dependence of bohm speed in presheath–sheath transition. *Physics of Plasmas*, 29(11): 113509, November 2022. doi: 10.1063/5.0110379. URL https://doi.org/10.1063/5.0110379.

[2] Robert E.H. Clark and Detlev H. Reiter, editors. *Nuclear Fusion Research*. Springer Berlin Heidelberg, 2005. doi: 10.1007/b138970. URL https://doi.org/10.1007/b138970.

[3] Ammar Hakim. The Gkeyll 2.0 Code: Documentation Home ; Gkeyll 2.0-alpha documentation — gkeyll.readthedocs.io. https://gkeyll.readthedocs.io/en/latest/index.html, 2016. [Accessed 03-08-2023].

[4] Irving Langmuir. Oscillations in ionized gases. *Proceedings of the National Academy of Sciences*, 14(8):627–637, August 1928. doi: 10.1073/pnas.14.8.627. URL https://doi.org/10.1073/pnas.14.8.627.

[5] Francis F. Chen. *Introduction to Plasma Physics and Controlled Fusion*. Springer International Publishing, 2016. doi: 10.1007/978-3-319-22309-4. URL https://doi.org/10.1007/978-3-319-22309-4.

[6] Irving Langmuir. Positive ion currents from the positive column of mercury arcs. *Science*, 58(1502):290–291, October 1923. doi: 10.1126/science.58.1502.290. URL https://doi.org/10.1126/science.58.1502.290.

[7] T. J. M. Boyd and J. J. Sanderson. *The Physics of Plasmas*. Cambridge University

Press, January 2003. doi: 10.1017/cbo9780511755750. URL https://doi.org/10.1017/cbo9780511755750.

[8] J. A. Bittencourt. *Fundamentals of Plasma Physics*. Springer New York, 2004. doi: 10.1007/978-1-4757-4030-1. URL https://doi.org/10.1007/978-1-4757-4030-1.

[9] K. U. Riemann. The bohm criterion and sheath formation. *Journal of Physics D: Applied Physics*, 24(4):493–518, April 1991. doi: 10.1088/0022-3727/24/4/001. URL https://doi.org/10.1088/0022-3727/24/4/001.

[10] R. N. Franklin. The plasma–sheath boundary region. *Journal of Physics D: Applied Physics*, 36(22):R309–R320, October 2003. doi: 10.1088/0022-3727/36/22/r01. URL https://doi.org/10.1088/0022-3727/36/22/r01.

[11] K.U. Riemann. The bohm criterion and boundary conditions for a multicomponent system. *IEEE Transactions on Plasma Science*, 23(4):709–716, 1995. doi: 10.1109/27.467993. URL https://doi.org/10.1109/27.467993.

[12] R. C. Bissell, P. C. Johnson, and P. C. Stangeby. A review of models for collisionless one-dimensional plasma flow to a boundary. *Physics of Fluids B: Plasma Physics*, 1(5):1133–1140, May 1989. doi: 10.1063/1.858983. URL https://doi.org/10.1063/1.858983.

[13] S. A. Self and H. N. Ewald. Static theory of a discharge column at intermediate pressures. *The Physics of Fluids*, 9(12):2486–2492, December 1966. doi: 10.1063/1.1761642. URL https://doi.org/10.1063/1.1761642.

[14] Robert Lewis Fullarton Boyd and J. B. Thompson. The operation of langmuir probes in electro-negative plasmas. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 252(1268):102–119, August 1959. doi: 10.1098/rspa.1959.0140. URL https://doi.org/10.1098/rspa.1959.0140.

[15] Scott D Baalrud, Brett Scheiner, Benjamin Yee, Matthew Hopkins, and Edward Barnat. Extensions and applications of the bohm criterion. *Plasma Physics and Controlled Fusion*, 57(4):044003, March 2015. doi: 10.1088/0741-3335/57/4/044003. URL https://doi.org/10.1088/0741-3335/57/4/044003.

[16] R. N. Franklin and J. R. Ockendon. Asymptotic matching of plasma and sheath in an active low pressure discharge. *Journal of Plasma Physics*, 4(2):371–385, May 1970. doi: 10.1017/s0022377800005067. URL https://doi.org/10.1017/s0022377800005067.

[17] R. N. Franklin. Where is the 'sheath edge'? *Journal of Physics D: Applied Physics*, 37(9):1342–1345, April 2004. doi: 10.1088/0022-3727/37/9/007. URL https://doi.org/10.1088/0022-3727/37/9/007.

[18] Xian-Zhu Tang. Kinetic magnetic dynamo in a sheath-limited high-temperature and low-density plasma. *Plasma Physics and Controlled Fusion*, 53(8):082002, May 2011. doi: 10.1088/0741-3335/53/8/082002. URL https://doi.org/10.1088/0741-3335/53/8/082002.

[19] Xian-Zhu Tang and Zehua Guo. Critical role of electron heat flux on bohm criterion. *Physics of Plasmas*, 23(12), December 2016. doi: 10.1063/1.4971808. URL https://doi.org/10.1063/1.4971808.

[20] R Chodura and F Pohl. Hydrodynamic equations for anisotropic plasmas in magnetic fields. II. transport equations including collisions. *Plasma Physics*, 13(8):645–658, August 1971. doi: 10.1088/0032-1028/13/8/003. URL https://doi.org/10.1088/0032-1028/13/8/003.

[21] S. I. Braginskii. Transport Processes in a Plasma. *Reviews of Plasma Physics*, 1:205, 1 1965. URL https://ui.adsabs.harvard.edu/abs/1965RvPP....1..205B.

[22] Geoffrey F. Chew, Marvin L. Goldberger, and Francis E. Low. The boltzmann equation an d the one-fluid hydromagnetic equations in the absence of particle collisions. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 236:112 – 118, 1956. URL https://api.semanticscholar.org/CorpusID:121756959.

[23] James Juno. A deep dive into the distribution function: Understanding phase space dynamics with continuum vlasov-maxwell simulations, 2020. URL https://arxiv.org/abs/2005.13539.

[24] D.R. Nicholson. *Introduction to Plasma Theory*. Wiley, 1983. ISBN 9780471090458. URL https://books.google.com/books?id=fyRRAAAAMAAJ.

[25] Gianpiero Colonna and Antonio D'Angola, editors. *Plasma Modeling*. 2053-2563. IOP Publishing, 2016. ISBN 978-0-7503-1200-4. doi: 10.1088/978-0-7503-1200-4. URL https://dx.doi.org/10.1088/978-0-7503-1200-4.

[26] Jan S. Hesthaven and Tim Warburton. *Nodal Discontinuous Galerkin Methods*. Springer New York, 2008. doi: 10.1007/978-0-387-72067-8. URL https://doi.org/10.1007/978-0-387-72067-8.

[27] Bruce Finlayson and L.E. Scriven. The method of weighted residuals - a review. *Appl. Mech. Rev.*, 19:735–748, 01 1966. URL https://www.researchgate.net/publication/284091948_The_method_of_weighted_residuals_-_A_review.

[28] W.J. Duncan, Aeronautical Research Council London (England), and Great Britain. Aeronautical Research Committee. *Galerkin's Method in Mechanics and Differential Equations*. ARC/R & M-1798. H.M. Stationery Office, 1937. URL https://books.google.com/books?id=37nsGwAACAAJ.

[29] Sergey Repin. One hundred years of the galerkin method. *Computational Methods in*

Applied Mathematics, 17(3):351–357, July 2017. doi: 10.1515/cmam-2017-0013. URL https://doi.org/10.1515/cmam-2017-0013.

[30] B. Cockburn, G.E. Karniadakis, and C.W. Shu. *Discontinuous Galerkin Methods: Theory, Computation and Applications*. Lecture Notes in Computational Science and Engineering. Springer Berlin Heidelberg, 2012. ISBN 9783642597213. URL https://books.google.com/books?id=uOHrCAAAQBAJ.

[31] Petr Cagas. Continuum kinetic simulations of plasma sheaths and instabilities, 2018. URL https://arxiv.org/abs/1809.06368.

[32] J. P. Dougherty. Model Fokker-Planck Equation for a Plasma and Its Solution. *The Physics of Fluids*, 7(11):1788–1799, 11 1964. ISSN 0031-9171. doi: 10.1063/1.2746779. URL https://doi.org/10.1063/1.2746779.

[33] Bhuvana Srinivasan, Petr Cagas, and Ammar H. Hakim. Simulations of plasma sheaths using continuum kinetic models. In *2016 IEEE International Conference on Plasma Science (ICOPS)*, pages 1–1, 2016. doi: 10.1109/PLASMA.2016.7534216. URL https://doi.org/10.1109/PLASMA.2016.7534216.

[34] T. Takizuka and H. Abe. A Binary Collision Model for Plasma Simulation with a Particle Code. *Journal of Computational Physics*, 25(3):205–219, November 1977. doi: 10.1016/0021-9991(77)90099-7. URL https://doi.org/10.1016/0021-9991(77)90099-7.

[35] E. P. Gross and M. Krook. Model for collision processes in gases: Small-amplitude oscillations of charged two-component systems. *Phys. Rev.*, 102:593–604, May 1956. doi: 10.1103/PhysRev.102.593. URL https://link.aps.org/doi/10.1103/PhysRev.102.593.

Appendices

Appendix A

Gkeyll Input File for Knudsen Number of 50 (lua)

```
-- Gkyl
-- 1X3V
--        c1    c2      c3      c4      c5      c6    c7      c8      c9      c10
     c11
-- N =  [7e20 4.4e20 3.6e20 8.8e20 4.4e20 2e20 5.7e20 4.4e20 2.8e20 1
   e19 2e19] m^-3
-- T =  [0.5  0.6     0.6     1.0     2.0     2.0   4.0     5.0     5.0     2.6
     6.0 ]  eV
-- Kn = [20    30      40      50      200     300   500     800     1000
   2000 5000]
local Plasma = require("App.PlasmaOnCartGrid").VlasovMaxwell()
local ep_0 = 8.854e-12 --Permittivity
local q_e = -1.6021766e-19 --Electron charge
local q_i = 1.6021766e-19 --Ion charge
local m_e = 9.109383e-31 --Electron Mass
local m_i = 1.6726218e-27 --Ion Mass
local n_e = 8.8e20 -- Electron density
local n_i = 8.8e20 -- Ion density
```

```
local T_e = 1*1.6021766e-19 -- Electron Temperature (J)
local T_i = 1*1.6021766e-19 -- Ion Temperature (J)
local ux_e = 0.00
local uy_e = 0.00
local uz_e = 0.00
local ux_i = 0.00
local uy_i = 0.00
local uz_i = 0.00
local u_B = math.sqrt(T_e/m_i) -- Bohm speed
local vt_e = math.sqrt(T_e/m_e) -- Electron thermal speed
local vt_i = math.sqrt(T_i/m_i) -- Ion thermal speed
local wp_e = math.sqrt(q_e*q_e*n_e/(ep_0*m_e))
local lambda_D = math.sqrt(ep_0*T_e/(n_e*q_e*q_e))
local mu_0 = 1.0/(ep_0*(10*vt_e)^2)
local lnl = 13.00
local pi = math.pi

local nu_ei = ((n_e)*q_e*q_e*q_e*q_e*lnl)/(16*pi*ep_0*ep_0*math.sqrt(
   m_e)*math.sqrt(T_e)*(T_e))
local nu_ee = nu_ei/math.sqrt(2)
local nu_ii = nu_ee*(vt_i/vt_e)
local nu_ie = nu_ee*(m_e/m_i)

local sourceLength = 32.0*lambda_D
local function maxwellian3D(n,ux,uy,uz,vt,vx,vy,vz)
    local v2 = (vx-ux)^2 + (vy-uy)^2 + (vz-uz)^2
    return (n/((2*vt*vt*math.pi)^(3/2)))*math.exp(-(v2)/(2*vt*vt))
```

```
41 end
42
43 sim = Plasma.App {
44     logToFile = false,
45     tEnd = 25000.0/wp_e,
46     nFrame = 100.0,
47     lower = {-128.0*lambda_D},
48     upper = {0},
49     cells = {128},
50     basis = "serendipity",
51     polyOrder = 2,
52     timeStepper = "rk3",
53     decompCuts = {32},
54     useShared = false,
55     periodicDirs = {},
56     elc = Plasma.Species {
57         charge = q_e,
58         mass = m_e,
59         lower = {-3.0*vt_e,-3.0*vt_e,-3.0*vt_e},
60         upper = {3.0*vt_e,3.0*vt_e,3.0*vt_e},
61         cells = {12,8,8},
62         decompCuts = {32,32,32},
63         init = function(t,xn)
64             local x,vx,vy,vz = xn[1],xn[2],xn[3],xn[4]
65             return maxwellian3D(n_e,ux_e,uy_e,uz_e,vt_e,vx,vy,vz)
66         end,
67         evolve = true,
```

```
        evolveCollisionless= true,
        bcx = { Plasma.AbsorbBC{},
                Plasma.ReflectBC{} },
        diagnostics = {"M0","M1i","M2","M2ij","intM0","Udrift","M2Flow"
           ,"M2Thermal","VtSq","intM1i","intM2","intM2Flow","
           intM2Thermal","M3i"},
        coll = Plasma.LBOCollisions {
            collideWith = {"elc","ion"},
            frequencies = {nu_ee,nu_ei},
    },
    src = Plasma.SteadySource {
        sourceSpecies = {"ion"},
        sourceLength = sourceLength,
        profile = function(t,xn)
            local x,vx,vy,vz = xn[1],xn[2],xn[3],xn[4]
            local m = maxwellian3D(1.0,ux_e,uy_e,uz_e,vt_e,vx,vy,vz)
            if (x > (-1)*sourceLength) and (x < (0)*sourceLength) then
                return m
            else
                return 0.0
            end
        end,
    },
},
ion = Plasma.Species{
    charge = q_i,
    mass = m_i,
```

```
93      lower = {-3.0*u_B,-3.0*u_B,-3.0*u_B},
94      upper = {3.0*u_B,3.0*u_B,3.0*u_B},
95      cells = {12,8,8},
96      decompCuts = {32,32,32},
97      init = function(t,xn)
98          local x,vx,vy,vz = xn[1],xn[2],xn[3],xn[4]
99          return maxwellian3D(n_i,ux_i,uy_i,uz_i,vt_i,vx,vy,vz)
100     end,
101     evolve = true,
102     evolveCollisionless = true,
103     bcx = { Plasma.AbsorbBC{},
104             Plasma.ReflectBC{} },
105     diagnostics = {"M0","M1i","M2","M2ij","intM0","Udrift","M2Flow","
           M2Thermal","VtSq","intM1i","intM2","intM2Flow","intM2Thermal","
           M3i"},
106     coll = Plasma.LBOCollisions {
107             collideWith = {"ion","elc"},
108             frequencies = {nu_ii,nu_ie},
109     },
110     src = Plasma.SteadySource {
111         sourceSpecies = {"ion"},
112         sourceLength = sourceLength,
113         profile = function(t,xn)
114             local x,vx,vy,vz = xn[1],xn[2],xn[3],xn[4]
115             local m = maxwellian3D(1.0,ux_i,uy_i,uz_i,vt_i,vx,vy,vz)
116             if (x > (-1)*sourceLength) and (x < (0)*sourceLength) then
117                 return m
```

```
            else
                return 0.0
            end
        end,
    },
},
field = Plasma.Field {
    epsilon0 = ep_0,
    mu0 = mu_0,
    init = function (t,xn)
        return 0.0,0.0,0.0,0.0,0.0,0.0
    end,
    evolve = true,
    bcx = { Plasma.ReflectBC{},
            Plasma.ReflectBC{} },
},
}
--run Application
sim:run()
```

Appendix B

Sheath Profile Computation (Jupyter notebook)

```
# import libraries
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import matplotlib.gridspec as gridspec
import matplotlib.colors as colors
import matplotlib.patches as patches
import postgkyl as pg
import scipy.integrate as integ
import scipy.optimize as opt
import scipy.special as spc
import scipy.fftpack as fft
from scipy.special import erf, comb, gamma, gammainc
import scipy.constants as const
from scipy.integrate import odeint
import sys
import math
```

```
19 from glob import glob
20 from mpl_toolkits.axes_grid1 import make_axes_locatable
21
22 # Calculate cell centered values from any edge values 1D array
23 def cell_center_values(cell_edge_values):
24     center = np.array([])
25     for i in range(len(cell_edge_values)-1):
26         avg = (cell_edge_values[i]+cell_edge_values[i+1])/2.0
27         center = np.append(center,avg)
28     return center
29
30 # define initial temps and densities and calculate initial debye length
       and thermal velocities
31 n0 = np.array([7e20,4.4e20,3.6e20,8.8e20,4.4e30,2e30,5.7e20,4.4e20,2.8
       e20,1e19,2e19])
32 T0 = np.array([0.5, 0.6,   0.6,   1,     2,     2,   4,     5,     5,
          2.6, 6.0 ])
33 T0 = T0*const.elementary_charge
34
35 lamda = np.zeros(len(n0))
36 vte0 = np.zeros(len(n0))
37 vti0 = np.zeros(len(n0))
38
39 for i in range(len(n0)):
40     lamda[i] = np.sqrt((const.epsilon_0*T0[i])/(n0[i]*const.
          elementary_charge**2))
41     vte0[i] = np.sqrt((T0[i])/const.electron_mass)
```

```
    vti0[i] = np.sqrt((T0[i])/const.proton_mass)

# change case and frame number for different collisional cases and
   different time frames
case = 4
index = case - 1
frame = 100

# M0
data = pg.GData('../sims/hpc/1x3v/c{}/c{}_ion_M0_{}.bp'.format(case,
   case,frame))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0),overwrite=True)
grid = data.getGrid()[0]
iM0 = data.getValues()[...,0]

data = pg.GData('../sims/hpc/1x3v/c{}/c{}_elc_M0_{}.bp'.format(case,
   case,frame))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0),overwrite=True)
eM0 = data.getValues()[...,0]

# M1
data = pg.GData('../sims/hpc/1x3v/c{}/c{}_ion_M1i_{}.bp'.format(case,
   case,frame))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0,1),overwrite=True)
```

```
65 iM1x = data.getValues()[..., 0]
66 iM1y = data.getValues()[..., 1]
67
68 data = pg.GData('../sims/hpc/1x3v/c{}/c{}_elc_M1i_{}.bp'.format(case,
       case,frame))
69 dg = pg.GInterpModal(data, 2, 'ms')
70 dg.interpolate((0,1),overwrite=True)
71 eM1x = data.getValues()[..., 0]
72 eM1y = data.getValues()[..., 1]
73
74 # M2
75 data = pg.GData('../sims/hpc/1x3v/c{}/c{}_ion_M2ij_{}.bp'.format(case,
       case,frame))
76 dg = pg.GInterpModal(data, 2, 'ms')
77 dg.interpolate((0,1,2,3),overwrite=True)
78 iM2xx = data.getValues()[..., 0]
79 iM2yy = data.getValues()[..., 3]
80
81 data = pg.GData('../sims/hpc/1x3v/c{}/c{}_elc_M2ij_{}.bp'.format(case,
       case,frame))
82 dg = pg.GInterpModal(data, 2, 'ms')
83 dg.interpolate((0,1,2,3),overwrite=True)
84 eM2xx = data.getValues()[..., 0]
85 eM2yy = data.getValues()[..., 3]
86
87 # Field
88 data = pg.GData('../sims/hpc/1x3v/c{}/c{}_field_{}.bp'.format(case,case
```

```
   ,frame))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0),overwrite=True)
Ex = data.getValues()[...,0]

# plots
# Densities
fig, ax = plt.subplots(figsize=(15,10))
ax.plot((grid[0:(len(grid)-1)]/lamda[index])+128,eM0/n0[index],'-',
   color='tab:blue',label='$n_e$',linewidth=5)
ax.plot((grid[0:(len(grid)-1)]/lamda[index])+128,iM0/n0[index],'-',
   color='tab:orange',label='$n_i$',linewidth=5)
ax.grid(linewidth=2)
ax.set_xlabel('$x/\lambda_D$',fontsize=30)
ax.set_ylabel('$n/n_0$',fontsize=30)
ax.legend(loc=2,ncol=1,fontsize=30)
ax.tick_params(axis='both', labelsize=30)
ax.set_xlim(-1,80)
ax.set_ylim(0,1.2)
ax.patch.set_edgecolor('black')
ax.patch.set_linewidth('2')
plt.tight_layout()
plt.savefig('c{}_density_CK.png'.format(case))
plt.show()

# Temperatures
eTx = const.electron_mass*((eM2xx/eM0)-(eM1x/eM0)**2)
```

```
eTy = const.electron_mass*((eM2yy/eM0)-(eM1y/eM0)**2)
iTx = const.proton_mass*((iM2xx/iM0)-(iM1x/iM0)**2)
iTy = const.proton_mass*((iM2yy/iM0)-(iM1y/iM0)**2)

fig, ax = plt.subplots(figsize=(15,10))
ax.plot((grid[0:(len(grid)-1)]/lamda[index])+128,eTx/T0[index],'-',
   color='tab:blue',label='$T_{ex}$',linewidth=5)
ax.plot((grid[0:(len(grid)-1)]/lamda[index])+128,eTy/T0[index],'-',
   color='tab:orange',label='$T_{ey}$',linewidth=5)
ax.plot((grid[0:(len(grid)-1)]/lamda[index])+128,iTx/T0[index],'-',
   color='tab:green',label='$T_{ix}$',linewidth=5)
ax.plot((grid[0:(len(grid)-1)]/lamda[index])+128,iTy/T0[index],'-',
   color='tab:red',label='$T_{iy}$',linewidth=5)
ax.grid(linewidth=2)
ax.set_xlabel('$x/\lambda_D$',fontsize=30)
ax.set_ylabel('$T/T0$',fontsize=30)
ax.legend(loc=4,ncol=2,fontsize=30)
ax.tick_params(axis='both', labelsize=30)
ax.set_xlim(-1,80)
# ax.set_ylim(0.1,0.8)
ax.patch.set_edgecolor('black')
ax.patch.set_linewidth('2')
plt.tight_layout()
plt.savefig('c{}_temperatures_CK.png'.format(case))
plt.show()

# Potential
```

```
136 phi = np.zeros(len(Ex))
137 dx = np.linspace(grid[0],grid[-1],len(phi))
138 phi =-integ.cumtrapz(Ex,dx,initial=0)
139
140 fig, ax = plt.subplots(figsize=(15,10))
141 ax.plot((grid[0:(len(grid)-1)]/lamda[index])+128,(const.
       elementary_charge*phi)/T0[index],color='tab:blue',linewidth=5)
142 plt.grid(linewidth=2)
143 ax.set_xlabel('$x/\lambda_D$',fontsize=30)
144 ax.set_ylabel('$e\phi/T_{e0}$',fontsize=30)
145 ax.tick_params(axis='both', labelsize=30)
146 ax.set_xlim(-1,80)
147 ax.set_ylim(-0.05,0.8)
148 ax.patch.set_edgecolor('black')
149 ax.patch.set_linewidth('2')
150 plt.tight_layout()
151 plt.savefig('c{}_potential_CK.png'.format(case))
152 plt.show()
153
154 # Ion speed
155 iux = iM1x/iM0
156 eux = eM1x/eM0
157 Cs = np.sqrt(((eTx)+(3*(iTx)))/const.proton_mass)
158
159 fig, ax = plt.subplots(figsize=(15,10))
160 ax.plot((grid[0:(len(grid)-1)]/lamda[index])+128,(iux/Cs),color='tab:
       blue',linewidth=5)
```

```
# ax.plot((grid/lamda[index])+128,(eux/Cs),color='tab:green',linewidth
   =5)
ax.grid(linewidth=2)
ax.set_xlabel('$x/\lambda_D$',fontsize=30)
ax.set_ylabel('$u_i/C_s$',fontsize=30)
ax.tick_params(axis='both', labelsize=30)
ax.set_xlim(-1,80)
ax.set_ylim(-2.1,-0.50)
ax.patch.set_edgecolor('black')
ax.patch.set_linewidth('2')
plt.tight_layout()
plt.savefig('c{}_ionSpeed_CK.png'.format(case))
plt.show()

# Charge density
abdif = abs(eM0-iM0)/abs(eM0+iM0)
dnedx = np.gradient(eM0,dx)
dnidx = np.gradient(iM0,dx)
reldif = abs(dnedx-dnidx)/abs(dnedx+dnidx)

fig, ax = plt.subplots(figsize=(15,10))
ax.plot((grid[0:(len(grid)-1)]/lamda[index])+128,abdif,'-',color='tab:
   blue',label=r'$\left|n_e-n_i\right| / \left|n_e+n_i\right|$',
   linewidth=5)
ax.plot((grid[0:(len(grid)-1)]/lamda[index])+128,reldif,'-',color='tab:
   orange',label=r'$\left| \frac{\partial n_e}{\partial x} - \frac{\
   partial n_i}{\partial x}\right| / \left|\frac{\partial n_e}{\partial
```

```
    x} + \frac{\partial n_i}{\partial x}\right|$',linewidth=5)
ax.grid(linewidth=2)
ax.set_xlabel('$x/\lambda_D$',fontsize=30)
ax.legend(loc=1,ncol=1,fontsize=30)
ax.tick_params(axis='both', labelsize=30)
ax.set_xlim(-0.25,20)
ax.set_ylim(-0.05,0.7)
ax.patch.set_edgecolor('black')
ax.patch.set_linewidth('2')
plt.tight_layout()
plt.savefig('c{}_chargeRatio_CK.png'.format(case))
plt.show()
```

Appendix C

Heat flux Computation (Jupyter notebook)

```
# import libraries
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import matplotlib.gridspec as gridspec
import matplotlib.colors as colors
import matplotlib.patches as patches
import postgkyl as pg
import scipy.integrate as integ
import scipy.optimize as opt
import scipy.special as spc
import scipy.fftpack as fft
from scipy.special import erf, comb, gamma, gammainc
import scipy.constants as const
from scipy.integrate import odeint
import sys
import math
```

```
from glob import glob
from mpl_toolkits.axes_grid1 import make_axes_locatable

# Calculate cell centered values from any edge values 1D array
def cell_center_values(cell_edge_values):
    center = np.array([])
    for i in range(len(cell_edge_values)-1):
        avg = (cell_edge_values[i]+cell_edge_values[i+1])/2.0
        center = np.append(center,avg)
    return center

# Initial temperatures and debye lengths
T = np.array([8.010883169999999e-20,9.613059803999999e
   -20,9.613059803999999e-20,1.602176634000000e-19,3.204353268000000e
   -19,3.204353268000000e-19,6.408706536000000e-19,8.010883170000000e
   -19,8.010883170000000e-19,4.165659248400000e-19,9.613059803999999e
   -19])
lamda = np.array([1.9868045698085713e-07,2.7451650119446644e
   -07,3.0348941107439841e-07,2.5059813351933860e-07,5.0119626703867719
   e-07,7.4339419947004640e-07,6.2274713169881040e
   -07,7.9246087930809712e-07,9.9340228490428551e-07,3.7905815293894535
   e-06,5.7583067084293684e-06])

# import distribution functions, M0, M1 and M2 for all the cases
frame = 100
case = 1
# Distribution functions
```

```
38 data = pg.GData('../sims/hpc/1x3v/c{}/c{}_elc_{}.bp'.format(case,case,
       frame))
39 dg = pg.GInterpModal(data, 2, 'ms')
40 dg.interpolate((0),overwrite=True)
41 c1_egrid = data.getGrid()
42 c1_edist = data.getValues()[...,0]
43 # Electron M0
44 data = pg.GData('../sims/hpc/1x3v/c{}/c{}_elc_M0_{}.bp'.format(case,
       case,frame))
45 dg = pg.GInterpModal(data, 2, 'ms')
46 dg.interpolate(overwrite=True)
47 c1_eM0 = data.getValues()[...,0]
48 # Electron M1i
49 data = pg.GData('../sims/hpc/1x3v/c{}/c{}_elc_M1i_{}.bp'.format(case,
       case,frame))
50 dg = pg.GInterpModal(data, 2, 'ms')
51 dg.interpolate((0),overwrite=True)
52 c1_eM1x = data.getValues()[..., 0]
53 # Electron M2ij
54 data = pg.GData('../sims/hpc/1x3v/c{}/c{}_elc_M2ij_{}.bp'.format(case,
       case,frame))
55 dg = pg.GInterpModal(data, 2, 'ms')
56 dg.interpolate((0),overwrite=True)
57 c1_eM2xx = data.getValues()[..., 0]
58
59 case = 4
60 # Distribution functions
```

```
data = pg.GData('../sims/hpc/1x3v/c{}/c{}_elc_{}.bp'.format(case,case,
    frame))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0),overwrite=True)
c4_egrid = data.getGrid()
c4_edist = data.getValues()[...,0]
# Electron M0
data = pg.GData('../sims/hpc/1x3v/c{}/c{}_elc_M0_{}.bp'.format(case,
    case,frame))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate(overwrite=True)
c4_eM0 = data.getValues()[...,0]
# Electron M1i
data = pg.GData('../sims/hpc/1x3v/c{}/c{}_elc_M1i_{}.bp'.format(case,
    case,frame))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0),overwrite=True)
c4_eM1x = data.getValues()[..., 0]
# Electron M2ij
data = pg.GData('../sims/hpc/1x3v/c{}/c{}_elc_M2ij_{}.bp'.format(case,
    case,frame))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0),overwrite=True)
c4_eM2xx = data.getValues()[..., 0]

case = 5
# Distribution functions
```

```
84 data = pg.GData('../sims/hpc/1x3v/c{}/c{}_elc_{}.bp'.format(case,case,
       frame))
85 dg = pg.GInterpModal(data, 2, 'ms')
86 dg.interpolate((0),overwrite=True)
87 c5_egrid = data.getGrid()
88 c5_edist = data.getValues()[...,0]
89 # Electron M0
90 data = pg.GData('../sims/hpc/1x3v/c{}/c{}_elc_M0_{}.bp'.format(case,
       case,frame))
91 dg = pg.GInterpModal(data, 2, 'ms')
92 dg.interpolate(overwrite=True)
93 c5_eM0 = data.getValues()[...,0]
94 # Electron M1i
95 data = pg.GData('../sims/hpc/1x3v/c{}/c{}_elc_M1i_{}.bp'.format(case,
       case,frame))
96 dg = pg.GInterpModal(data, 2, 'ms')
97 dg.interpolate((0),overwrite=True)
98 c5_eM1x = data.getValues()[..., 0]
99 # Electron M2ij
100 data = pg.GData('../sims/hpc/1x3v/c{}/c{}_elc_M2ij_{}.bp'.format(case,
       case,frame))
101 dg = pg.GInterpModal(data, 2, 'ms')
102 dg.interpolate((0),overwrite=True)
103 c5_eM2xx = data.getValues()[..., 0]
104
105 case = 7
106 # Distribution functions
```

```
107 data = pg.GData('../sims/hpc/1x3v/c{}/c{}_elc_{}.bp'.format(case,case,
    frame))
108 dg = pg.GInterpModal(data, 2, 'ms')
109 dg.interpolate((0),overwrite=True)
110 c7_egrid = data.getGrid()
111 c7_edist = data.getValues()[...,0]
112 # Electron M0
113 data = pg.GData('../sims/hpc/1x3v/c{}/c{}_elc_M0_{}.bp'.format(case,
    case,frame))
114 dg = pg.GInterpModal(data, 2, 'ms')
115 dg.interpolate(overwrite=True)
116 c7_eM0 = data.getValues()[...,0]
117 # Electron M1i
118 data = pg.GData('../sims/hpc/1x3v/c{}/c{}_elc_M1i_{}.bp'.format(case,
    case,frame))
119 dg = pg.GInterpModal(data, 2, 'ms')
120 dg.interpolate((0),overwrite=True)
121 c7_eM1x = data.getValues()[..., 0]
122 # Electron M2ij
123 data = pg.GData('../sims/hpc/1x3v/c{}/c{}_elc_M2ij_{}.bp'.format(case,
    case,frame))
124 dg = pg.GInterpModal(data, 2, 'ms')
125 dg.interpolate((0),overwrite=True)
126 c7_eM2xx = data.getValues()[..., 0]
127
128 case = 10
129 # Distribution functions
```

```
data = pg.GData('../sims/hpc/1x3v/c{}/c{}_elc_{}.bp'.format(case,case,
    frame))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0),overwrite=True)
c10_egrid = data.getGrid()
c10_edist = data.getValues()[...,0]
# Electron M0
data = pg.GData('../sims/hpc/1x3v/c{}/c{}_elc_M0_{}.bp'.format(case,
    case,frame))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate(overwrite=True)
c10_eM0 = data.getValues()[...,0]
# Electron M1i
data = pg.GData('../sims/hpc/1x3v/c{}/c{}_elc_M1i_{}.bp'.format(case,
    case,frame))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0),overwrite=True)
c10_eM1x = data.getValues()[..., 0]
# Electron M2ij
data = pg.GData('../sims/hpc/1x3v/c{}/c{}_elc_M2ij_{}.bp'.format(case,
    case,frame))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0),overwrite=True)
c10_eM2xx = data.getValues()[..., 0]

# Calculate the heat flux for each case
c1_eux = c1_eM1x/c1_eM0                    #Drift velocity in each
```

```
   spatial cell
c1_evx = cell_center_values(c1_egrid[1])
c1_eveldiff = np.zeros((384,36))
for i in range(384):
    c1_eveldiff[i] = c1_evx-c1_eux[i]     #compute vx-ux
c1_eveldiff3 = c1_eveldiff**3             #compute (vx-ux)^3
c1_efvdiff = c1_edist.copy()
for i in range(384):
    for j in range(36):
        c1_efvdiff[i,j,:] = c1_eveldiff3[i,j]*c1_efvdiff[i,j,:] #
           perform (vx-ux)^3 times f
# Integration
c1_edvx = c1_egrid[1][1]-c1_egrid[1][0]
c1_edvy = c1_egrid[2][1]-c1_egrid[2][0]
c1_edvz = c1_egrid[3][1]-c1_egrid[3][0]
c1_Iz = np.trapz((c1_efvdiff*c1_edvz),axis=3)
c1_Iy = np.trapz((c1_Iz*c1_edvy),axis=2)
c1_I = np.trapz((c1_Iy*c1_edvx),axis=1)
c1_qx = const.electron_mass*c1_I
# Normalization
c1_Tex = const.electron_mass*((c1_eM2xx/c1_eM0)-(c1_eux)**2)
# ------------------------------------------------------------
c4_eux = c4_eM1x/c4_eM0
c4_evx = cell_center_values(c4_egrid[1])
c4_eveldiff = np.zeros((384,36))
for i in range(384):
    c4_eveldiff[i] = c4_evx-c4_eux[i]
```

```
c4_eveldiff3 = c4_eveldiff**3
c4_efvdiff = c4_edist.copy()
for i in range(384):
    for j in range(36):
        c4_efvdiff[i,j,:] = c4_eveldiff3[i,j]*c4_efvdiff[i,j,:]
# Integration
c4_edvx = c4_egrid[1][1]-c4_egrid[1][0]
c4_edvy = c4_egrid[2][1]-c4_egrid[2][0]
c4_edvz = c4_egrid[3][1]-c4_egrid[3][0]
c4_Iz = np.trapz((c4_efvdiff*c4_edvz),axis=3)
c4_Iy = np.trapz((c4_Iz*c4_edvy),axis=2)
c4_I = np.trapz((c4_Iy*c4_edvx),axis=1)
c4_qx = const.electron_mass*c4_I
# Normalization
c4_Tex = const.electron_mass*((c4_eM2xx/c4_eM0)-(c4_eux)**2)
# ------------------------------------------------------------
c5_eux = c5_eM1x/c5_eM0
c5_evx = cell_center_values(c5_egrid[1])
c5_eveldiff = np.zeros((384,54))
for i in range(384):
    c5_eveldiff[i] = c5_evx-c5_eux[i]
c5_eveldiff3 = c5_eveldiff**3
c5_efvdiff = c5_edist.copy()
for i in range(384):
    for j in range(54):
        c5_efvdiff[i,j,:] = c5_eveldiff3[i,j]*c5_efvdiff[i,j,:]
# Integration
```

```
c5_edvx = c5_egrid[1][1]-c5_egrid[1][0]
c5_edvy = c5_egrid[2][1]-c5_egrid[2][0]
c5_edvz = c5_egrid[3][1]-c5_egrid[3][0]
c5_Iz = np.trapz((c5_efvdiff*c5_edvz),axis=3)
c5_Iy = np.trapz((c5_Iz*c5_edvy),axis=2)
c5_I = np.trapz((c5_Iy*c5_edvx),axis=1)
c5_qx = const.electron_mass*c5_I
# Normalization
c5_Tex = const.electron_mass*((c5_eM2xx/c5_eM0)-(c5_eux)**2)
# ------------------------------------------------------------
c7_eux = c7_eM1x/c7_eM0
c7_evx = cell_center_values(c7_egrid[1])
c7_eveldiff = np.zeros((384,54))
for i in range(384):
    c7_eveldiff[i] = c7_evx-c7_eux[i]
c7_eveldiff3 = c7_eveldiff**3
c7_efvdiff = c7_edist.copy()
for i in range(384):
    for j in range(54):
        c7_efvdiff[i,j,:] = c7_eveldiff3[i,j]*c7_efvdiff[i,j,:]
# Integration
c7_edvx = c7_egrid[1][1]-c7_egrid[1][0]
c7_edvy = c7_egrid[2][1]-c7_egrid[2][0]
c7_edvz = c7_egrid[3][1]-c7_egrid[3][0]
c7_Iz = np.trapz((c7_efvdiff*c7_edvz),axis=3)
c7_Iy = np.trapz((c7_Iz*c7_edvy),axis=2)
c7_I = np.trapz((c7_Iy*c7_edvx),axis=1)
```

```
c7_qx = const.electron_mass*c7_I
# Normalization
c7_Tex = const.electron_mass*((c7_eM2xx/c7_eM0)-(c7_eux)**2)
# ------------------------------------------------------------
c10_eux = c10_eM1x/c10_eM0
c10_evx = cell_center_values(c10_egrid[1])
c10_eveldiff = np.zeros((384,54))
for i in range(384):
    c10_eveldiff[i] = c10_evx-c10_eux[i]
c10_eveldiff3 = c10_eveldiff**3
c10_efvdiff = c10_edist.copy()
for i in range(384):
    for j in range(54):
        c10_efvdiff[i,j,:] = c10_eveldiff3[i,j]*c10_efvdiff[i,j,:]
# Integration
c10_edvx = c10_egrid[1][1]-c10_egrid[1][0]
c10_edvy = c10_egrid[2][1]-c10_egrid[2][0]
c10_edvz = c10_egrid[3][1]-c10_egrid[3][0]
c10_Iz = np.trapz((c10_efvdiff*c10_edvz),axis=3)
c10_Iy = np.trapz((c10_Iz*c10_edvy),axis=2)
c10_I = np.trapz((c10_Iy*c10_edvx),axis=1)
c10_qx = const.electron_mass*c10_I
# Normalization
c10_Tex = const.electron_mass*((c10_eM2xx/c10_eM0)-(c10_eux)**2)

# Heat Flux plot
fig, ax = plt.subplots(figsize=(15,10))
```

```
259
260 ax.plot((c1_egrid[0][0:(len(c1_egrid[0])-1)]/lamda[0])+128,(c1_qx/(
       c1_eM1x*c1_Tex)),'-',color='tab:blue',label='Kn = 20',linewidth=5)
261 ax.plot((c4_egrid[0][0:(len(c4_egrid[0])-1)]/lamda[3])+128,(c4_qx/(
       c4_eM1x*c4_Tex)),'--',color='tab:purple',label='Kn = 50',linewidth
       =5)
262 ax.plot((c5_egrid[0][0:(len(c5_egrid[0])-1)]/lamda[4])+128,(c5_qx/(
       c5_eM1x*c5_Tex)),'-',color='tab:orange',label='Kn = 200',linewidth
       =5)
263 ax.plot((c7_egrid[0][0:(len(c7_egrid[0])-1)]/lamda[6])+128,(c7_qx/(
       c7_eM1x*c7_Tex)),'-',color='tab:green',label='Kn = 500',linewidth=5)
264 ax.plot((c10_egrid[0][0:(len(c10_egrid[0])-1)]/lamda[9])+128,(c10_qx/(
       c10_eM1x*c10_Tex)),'-',color='tab:red',label='Kn = 2000',linewidth
       =5)
265 ax.grid(linewidth=2)
266 ax.set_xlim(0,20)
267 # ax.set_ylim(1.25,4.5)
268 ax.set_xlabel('$x/\lambda_D$',fontsize=25)
269 ax.set_ylabel('$q_{n}^{e}/(\Gamma_{e}T_{ex})$',fontsize=30)
270 ax.legend(loc=3,ncol=2,fontsize=30)
271 ax.tick_params(axis='both', labelsize=30)
272 ax.patch.set_edgecolor('black')
273 ax.patch.set_linewidth('2')
274 plt.tight_layout()
275 plt.savefig('Heatflux_CK.png')
276 plt.show()
```

Appendix D

Momentum, Energy and Bohm Speed Computation using LBO (Jupyter notebook)

```
# import libraries
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import matplotlib.gridspec as gridspec
import matplotlib.colors as colors
import matplotlib.patches as patches
import postgkyl as pg
import scipy.integrate as integ
import scipy.optimize as opt
import scipy.special as spc
import scipy.fftpack as fft
from scipy.special import erf, comb, gamma, gammainc
import scipy.constants as const
from scipy.integrate import odeint
```

```
import sys
import math
from glob import glob
from mpl_toolkits.axes_grid1 import make_axes_locatable

# Calculate cell centered values from any edge values 1D array
def cell_center_values(cell_edge_values):
    center = np.array([])
    for i in range(len(cell_edge_values)-1):
        avg = (cell_edge_values[i]+cell_edge_values[i+1])/2.0
        center = np.append(center,avg)
    return center

# Initial density and temperatures to calculate debye length and
   thermal velocities
n0 = np.array([7e20,4.4e20,3.6e20,8.8e20,4.4e20,2e20,5.7e20,4.4e20,2.8
   e20,1e19,1e19])
T0 = np.array([0.5, 0.6,   0.6,   1,     2,     2,   4,     5,     5,
       2.6, 6.0 ])
T0 = T0*const.elementary_charge

lamda = np.zeros(len(n0))
vte0 = np.zeros(len(n0))
vti0 = np.zeros(len(n0))

for i in range(len(n0)):
    lamda[i] = np.sqrt((const.epsilon_0*T0[i])/(n0[i]*const.
```

```
       elementary_charge**2))
    vte0[i] = np.sqrt((T0[i])/const.electron_mass)
    vti0[i] = np.sqrt((T0[i])/const.proton_mass)

#Specify case and time frame
frame = 100
case = 4

#Import gkeyll output files in each case
# Distribution function
data = pg.GData('c{}_elc_{}.bp'.format(case,100))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0),overwrite=True)
egrid = data.getGrid()
edist = data.getValues()[...,0]
data = pg.GData('c{}_ion_{}.bp'.format(case,100))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0),overwrite=True)
igrid = data.getGrid()
idist = data.getValues()[...,0]
# M0
data = pg.GData('c{}_elc_M0_{}.bp'.format(case,100))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate(overwrite=True)
eM0 = data.getValues()[...,0]
data = pg.GData('c{}_ion_M0_{}.bp'.format(case,100))
dg = pg.GInterpModal(data, 2, 'ms')
```

```
dg.interpolate(overwrite=True)
iM0 = data.getValues()[...,0]
# M1i
data = pg.GData('c{}_elc_M1i_{}.bp'.format(case,100))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0,1,2),overwrite=True)
eM1x = data.getValues()[..., 0]
eM1y = data.getValues()[..., 1]
eM1z = data.getValues()[..., 2]
data = pg.GData('c{}_ion_M1i_{}.bp'.format(case,100))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0,1,2),overwrite=True)
iM1x = data.getValues()[..., 0]
iM1y = data.getValues()[..., 1]
iM1z = data.getValues()[..., 2]
# M2ij
data = pg.GData('c{}_elc_M2ij_{}.bp'.format(case,100))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0,1,2,3,4,5),overwrite=True)
eM2xx = data.getValues()[..., 0]
eM2yy = data.getValues()[..., 3]
eM2zz = data.getValues()[..., 5]
data = pg.GData('c{}_ion_M2ij_{}.bp'.format(case,100))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0,1,2,3,4,5),overwrite=True)
iM2xx = data.getValues()[..., 0]
iM2yy = data.getValues()[..., 3]
```

```
iM2zz = data.getValues()[..., 5]
# Udrift
data = pg.GData('c{}_elc_Udrift_{}.bp'.format(case,100))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0,1,2),overwrite=True)
eUdrift = data.getValues()
eux = data.getValues()[..., 0]
euy = data.getValues()[..., 1]
euz = data.getValues()[..., 2]
data = pg.GData('c{}_ion_Udrift_{}.bp'.format(case,100))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0,1,2),overwrite=True)
iUdrift = data.getValues()
iux = data.getValues()[..., 0]
iuy = data.getValues()[..., 1]
iuz = data.getValues()[..., 2]
# Field
data = pg.GData('c{}_field_{}.bp'.format(case,frame))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0),overwrite=True)
Ex = data.getValues()[...,0]

#since cases are run with different resolution
if case == 7:
    xres = 72
    yres = 48
    zres = 48
```

```
elif case == 10:
    xres = 54
    yres = 36
    zres = 36
elif case == 9:
    xres = 54
    yres = 36
    zres = 36
elif case == 5:
    xres = 54
    yres = 36
    zres = 36
else:
    xres = 36
    yres = 24
    zres = 24

# Calculate electron and ion heat flux
eux = eM1x/eM0
evx = cell_center_values(egrid[1])
eveldiff = np.zeros((384,xres))
for i in range(384):
    eveldiff[i] = evx-eux[i]
eveldiff3 = eveldiff**3
efvdiff = edist.copy()
for i in range(384):
    for j in range(xres):
```

```
148         efvdiff[i,j,:] = eveldiff3[i,j]*efvdiff[i,j,:]
149 # Integration
150 edvx = egrid[1][1]-egrid[1][0]
151 edvy = egrid[2][1]-egrid[2][0]
152 edvz = egrid[3][1]-egrid[3][0]
153 Iz = np.trapz((efvdiff*edvz),axis=3)
154 Iy = np.trapz((Iz*edvy),axis=2)
155 I = np.trapz((Iy*edvx),axis=1)
156 qx = const.electron_mass*I
157
158 iux = iM1x/iM0
159 ivx = cell_center_values(igrid[1])
160 iveldiff = np.zeros((384,xres))
161 for i in range(384):
162     iveldiff[i] = ivx-iux[i]
163 iveldiff3 = iveldiff**3
164 ifvdiff = idist.copy()
165 for i in range(384):
166     for j in range(xres):
167         ifvdiff[i,j,:] = iveldiff3[i,j]*ifvdiff[i,j,:]
168 # Integration
169 idvx = igrid[1][1]-igrid[1][0]
170 idvy = igrid[2][1]-igrid[2][0]
171 idvz = igrid[3][1]-igrid[3][0]
172 iIz = np.trapz((ifvdiff*idvz),axis=3)
173 iIy = np.trapz((iIz*idvy),axis=2)
174 iI =  np.trapz((iIy*idvx),axis=1)
```

```
iqx = const.proton_mass*iI

# calculate collision integral and isotropization from LBO operator
me = const.electron_mass
mi = const.proton_mass
pi = math.pi
e = const.elementary_charge
ep0 = const.epsilon_0

# Temperatures
Tex = me*((eM2xx/eM0)-(eux)**2)
Tey = me*((eM2yy/eM0)-(euy)**2)
Tez = me*((eM2zz/eM0)-(euz)**2)

Tix = mi*((iM2xx/iM0)-(iux)**2)
Tiy = mi*((iM2yy/iM0)-(iuy)**2)
Tiz = mi*((iM2zz/iM0)-(iuz)**2)

Vt2_e = (Tex+Tey+Tez)/(3*me)
Vt2_i = (Tix+Tiy+Tiz)/(3*mi)

#anisotropic Collision frequencies
nu_ei = (np.sqrt(pi)*iM0*(e**4)*13)/(np.sqrt(2)*((4*pi*ep0)**2)*np.sqrt
   (me*Tex)*Tey)
nu_ee = (eM0*nu_ei)/(iM0*np.sqrt(2))
nu_ie = nu_ee*(me/mi)*(eM0/iM0)
nu_ii = (nu_ee)*(iM0/iM0)*(Tey/Tiy)*(np.sqrt(me*Tex)/np.sqrt(mi*Tix))
```

```
201
202 #LBO terms
203 delta_ei = (2*me*eM0*nu_ei)/((me*eM0*nu_ei)+(mi*iM0*nu_ie))
204 alpha_ei = (me*eM0*nu_ei*delta_ei)/(me+mi)
205
206 Udiff = eUdrift-iUdrift
207 Udiff_dot = np.einsum('ij,ij->i',Udiff,Udiff)
208
209 Vt2_ei = (Vt2_e) + ((alpha_ei/2) * ((me+mi)/(me*eM0*nu_ei)) * (1/(1+(me
    /mi))) * ( (Vt2_i) - ((me/mi)*Vt2_e) + ((Udiff_dot)/3) ) ) #checked
210 U_ei   = (eUdrift) + ((alpha_ei[:,np.newaxis]/2) * ((me+mi)/(me*eM0[:,
    np.newaxis]*nu_ei[:,np.newaxis])) * (iUdrift-eUdrift)) #checked
211
212 # -----
213
214 delta_ee = (2*me*eM0*nu_ee)/((me*eM0*nu_ee)+(me*eM0*nu_ee))
215 alpha_ee = (me*eM0*nu_ee*delta_ee)/(me+me)
216
217 Udiff = eUdrift-eUdrift
218 Udiff_dot = np.einsum('ij,ij->i',Udiff,Udiff)
219
220 Vt2_ee = (Vt2_e) + ((alpha_ee/2) * ((me+me)/(me*eM0*nu_ee)) * (1/(1+(me
    /me))) * ( (Vt2_e) - ((me/me)*Vt2_e) + ((Udiff_dot)/3) ) ) #checked
221 U_ee   = (eUdrift) + ((alpha_ee[:,np.newaxis]/2) * ((me+me)/(me*eM0[:,
    np.newaxis]*nu_ee[:,np.newaxis])) * (eUdrift-eUdrift)) #checked
222
223 # -----
```

```

delta_ii = (2*mi*iM0*nu_ii)/((mi*iM0*nu_ii)+(mi*iM0*nu_ii))
alpha_ii = (mi*iM0*nu_ii*delta_ii)/(mi+mi)

Udiff = iUdrift-iUdrift
Udiff_dot = np.einsum('ij,ij->i',Udiff,Udiff)

Vt2_ii = (Vt2_i) + ((alpha_ii/2) * ((mi+mi)/(mi*iM0*nu_ii)) * (1/(1+(mi
    /mi))) * ( (Vt2_i) - ((mi/mi)*Vt2_i) + ((Udiff_dot)/3) ) ) #checked
U_ii   = (iUdrift) + ((alpha_ii[:,np.newaxis]/2) * ((mi+mi)/(mi*iM0[:,
    np.newaxis]*nu_ii[:,np.newaxis])) * (iUdrift-iUdrift)) #checked

# Compute Qei

vx = cell_center_values(egrid[1])
vy = cell_center_values(egrid[2])
vz = cell_center_values(egrid[3])

dfedvx = np.gradient(edist,vx,axis=1)
dfedvy = np.gradient(edist,vy,axis=2)
dfedvz = np.gradient(edist,vz,axis=3)

# Compute second term
vteidfdx = np.zeros_like(dfedvx)
vteidfdy = np.zeros_like(dfedvy)
vteidfdz = np.zeros_like(dfedvz)

```

```
249 for i in range(384):
250     vteidfdx[i]=Vt2_ei[i]*dfedvx[i]
251     vteidfdy[i]=Vt2_ei[i]*dfedvy[i]
252     vteidfdz[i]=Vt2_ei[i]*dfedvz[i]
253 
254 # Compute first term
255 vxuei = np.zeros((384,xres))
256 vyuei = np.zeros((384,yres))
257 vzuei = np.zeros((384,zres))
258 
259 for i in range(384):
260     vxuei[i]=vx-U_ei[i][0]
261     vyuei[i]=vy-U_ei[i][1]
262     vzuei[i]=vz-U_ei[i][2]
263 
264 fvueix = np.zeros_like(edist)
265 fvueiy = np.zeros_like(edist)
266 fvueiz = np.zeros_like(edist)
267 
268 for i in range(384):
269     for j1 in range(xres):
270         fvueix[i,j1,:,:]=vxuei[i,j1]*edist[i,j1,:,:]
271     for j2 in range(yres):
272         fvueiy[i,:,j2,:]=vyuei[i,j2]*edist[i,:,j2,:]
273         fvueiz[i,:,:,j2]=vzuei[i,j2]*edist[i,:,:,j2]
274 
275 # Add the terms
```

```
276 phieix = fvueix + vteidfdx
277 phieiy = fvueiy + vteidfdy
278 phieiz = fvueiz + vteidfdz
279
280 dphieidvx = np.gradient(phieix,vx,axis=1)
281 dphieidvy = np.gradient(phieiy,vy,axis=2)
282 dphieidvz = np.gradient(phieiz,vz,axis=3)
283
284 divphiei = dphieidvx+dphieidvy+dphieidvz
285
286 dfeidt = np.zeros_like(divphiei)
287 for i in range(384):
288     dfeidt[i,:] = nu_ei[i]*divphiei[i,:] #this is the collision
        integral due to electron-ion collisions
289
290 # Qee
291 # Compute second term
292 vteedfdx = np.zeros_like(dfedvx)
293 vteedfdy = np.zeros_like(dfedvy)
294 vteedfdz = np.zeros_like(dfedvz)
295
296 for i in range(384):
297     vteedfdx[i]=Vt2_ee[i]*dfedvx[i]
298     vteedfdy[i]=Vt2_ee[i]*dfedvy[i]
299     vteedfdz[i]=Vt2_ee[i]*dfedvz[i]
300
301 # Compute first term
```

```
vxuee = np.zeros((384,xres))
vyuee = np.zeros((384,yres))
vzuee = np.zeros((384,zres))

for i in range(384):
    vxuee[i]=vx-U_ee[i][0]
    vyuee[i]=vy-U_ee[i][1]
    vzuee[i]=vz-U_ee[i][2]

fvueex = np.zeros_like(edist)
fvueey = np.zeros_like(edist)
fvueez = np.zeros_like(edist)

for i in range(384):
    for j1 in range(xres):
        fvueex[i,j1,:,:]=vxuee[i,j1]*edist[i,j1,:,:]
    for j2 in range(yres):
        fvueey[i,:,j2,:]=vyuee[i,j2]*edist[i,:,j2,:]
        fvueez[i,:,:,j2]=vzuee[i,j2]*edist[i,:,:,j2]

# Add the terms
phieex = fvueex + vteedfdx
phieey = fvueey + vteedfdy
phieez = fvueez + vteedfdz

dphieedvx = np.gradient(phieex,vx,axis=1)
dphieedvy = np.gradient(phieey,vy,axis=2)
```

```
dphieedvz = np.gradient(phieez,vz,axis=3)

divphiee = dphieedvx+dphieedvy+dphieedvz

dfeedt = np.zeros_like(divphiee)
for i in range(384):
    dfeedt[i,:] = nu_ee[i]*divphiee[i,:] #this is the collision
        integral due to electron-electron collisions

vxmux = np.zeros((384,xres))

for i in range(384):
    vxmux[i] = vx-eux[i]

vxmux2 =  vxmux**2

vxmuxdfeidt = np.zeros_like(edist)
vxmuxdfeedt = np.zeros_like(edist)

for i in range(384):
    for j in range(xres):
        vxmuxdfeidt[i,j,:] = vxmux2[i,j]*dfeidt[i,j,:]
        vxmuxdfeedt[i,j,:] = vxmux2[i,j]*dfeedt[i,j,:]

dvx = vx[1]-vx[0]
dvy = vy[1]-vy[0]
dvz = vz[1]-vz[0]
```

```
355
356 Izei = np.trapz((vxmuxdfeidt*dvz),axis=3) # --NEW--
357 Iyei = np.trapz((Izei*dvy),axis=2)
358 Ixei = np.trapz((Iyei*dvx),axis=1)
359 Qei = Ixei*me #Qei computed
360
361 Izee = np.trapz((vxmuxdfeedt*dvz),axis=3) # --NEW--
362 Iyee = np.trapz((Izee*dvy),axis=2)
363 Ixee = np.trapz((Iyee*dvx),axis=1)
364 Qee  = Ixee*me #Qee computed
365
366 # Qii
367
368 vx = cell_center_values(igrid[1])
369 vy = cell_center_values(igrid[2])
370 vz = cell_center_values(igrid[3])
371
372 dfidvx = np.gradient(idist,vx,axis=1)
373 dfidvy = np.gradient(idist,vy,axis=2)
374 dfidvz = np.gradient(idist,vz,axis=3)
375
376 # Compute second term
377 vtiidfdx = np.zeros_like(dfidvx)
378 vtiidfdy = np.zeros_like(dfidvy)
379 vtiidfdz = np.zeros_like(dfidvz)
380
381 for i in range(384):
```

```
    vtiidfdx[i]=Vt2_ii[i]*dfidvx[i]
    vtiidfdy[i]=Vt2_ii[i]*dfidvy[i]
    vtiidfdz[i]=Vt2_ii[i]*dfidvz[i]

# Compute first term
vxuii = np.zeros((384,xres))
vyuii = np.zeros((384,yres))
vzuii = np.zeros((384,zres))

for i in range(384):
    vxuii[i]=vx-U_ii[i][0]
    vyuii[i]=vy-U_ii[i][1]
    vzuii[i]=vz-U_ii[i][2]

fvuiix = np.zeros_like(idist)
fvuiiy = np.zeros_like(idist)
fvuiiz = np.zeros_like(idist)

for i in range(384):
    for j1 in range(xres):
        fvuiix[i,j1,:,:]=vxuii[i,j1]*idist[i,j1,:,:]
    for j2 in range(yres):
        fvuiiy[i,:,j2,:]=vyuii[i,j2]*idist[i,:,j2,:]
        fvuiiz[i,:,:,j2]=vzuii[i,j2]*idist[i,:,:,j2]

# Add the terms
phiiix = fvuiix + vtiidfdx
```

```
phiiiy = fvuiiy + vtiidfdy
phiiiz = fvuiiz + vtiidfdz

dphiiidvx = np.gradient(phiiix,vx,axis=1)
dphiiidvy = np.gradient(phiiiy,vy,axis=2)
dphiiidvz = np.gradient(phiiiz,vz,axis=3)

divphiii = dphiiidvx+dphiiidvy+dphiiidvz

dfiidt = np.zeros_like(divphiii)
for i in range(384):
    dfiidt[i,:] = nu_ii[i]*divphiii[i,:]

vxmux = np.zeros((384,xres))

for i in range(384):
    vxmux[i] = vx-iux[i]

vxmux2 =  vxmux**2

vxmuxdfiidt = np.zeros_like(idist)

for i in range(384):
    for j in range(xres):
        vxmuxdfiidt[i,j,:] = vxmux2[i,j]*dfiidt[i,j,:]

dvx = vx[1]-vx[0]
```

```
dvy = vy[1]-vy[0]
dvz = vz[1]-vz[0]

Izii = np.trapz((vxmuxdfiidt*dvz),axis=3) # --NEW--
Iyii = np.trapz((Izii*dvy),axis=2)
Ixii = np.trapz((Iyii*dvx),axis=1)
Qii = Ixii*mi

# Energy plot

x = cell_center_values(egrid[0])
dTexdx = np.gradient(Tex,x)
deuxdx = np.gradient(eux,x)

EnergyLHS = (eM0*eux*dTexdx) + (2*eM0*Tex*deuxdx)

dqedx = -1*(np.gradient(qx,x))

EnergyISO = (EnergyLHS) - (Qee) - (Qei)

fig, ax = plt.subplots(figsize=(15,10))
ax.plot((egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128, ((dqedx)
   /(1.6982e13)), label = '$-(\partial q_n^e/\partial x)$' ,color='tab:
   blue',linewidth=5)
ax.plot((egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128, ((EnergyISO)
   /(1.6982e13)), label = '$ n_e u_{ex} (\partial T_{ex}/\partial x)+2
   n_eT_{ex}(\partial u_{ex}/\partial x) - Q_{ee} -Q_{ei} $' ,color='
```

```
    tab:orange',linewidth=5)
ax.plot((egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128, ((EnergyLHS)
    /1.6982e13), label = '$ n_e u_{ex} (\partial T_{ex}/\partial x)+2
    n_eT_{ex}(\partial u_{ex}/\partial x) $' ,color='tab:green',
    linewidth=5)
ax.grid(linewidth=2)
ax.set_xlabel('$x/\lambda_D$',fontsize=30)
ax.set_xlim(-1,80)
ax.set_ylim(-0.025,0.175)
ax.legend(loc=1,ncol=1,fontsize=30)
ax.tick_params(axis='both', labelsize=30)
ax.patch.set_edgecolor('black')
ax.patch.set_linewidth('2')
plt.tight_layout()
plt.savefig('c{}_eenergy_LBO.png'.format(case))
plt.show()

# Thermal force

vdfdt = np.zeros_like(edist)

for i in range(384):
    for j in range(xres):
        vdfdt[i,j,:] = vx[j]*dfeidt[i,j,:]

dvx = vx[1]-vx[0]
dvy = vy[1]-vy[0]
```

```
dvz = vz[1]-vz[0]
Iz = np.trapz((vdfdt*dvz),axis=3) # --NEW--
Iy = np.trapz((Iz*dvy),axis=2)
I  = np.trapz((Iy*dvx),axis=1)
Rt = I*me

alpha = -Rt/(eM0*dTexdx)

#Momentum plot

Pex = eM0*Tex
dpexdx = np.gradient(Pex,x)

emom = (eM0*me*eux*deuxdx)+(dpexdx)+(e*eM0*Ex)
Rt_lim = (-1*0.71*eM0*dTexdx)

fig, ax = plt.subplots(figsize=(15,10))
ax.plot( (egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128, (Rt)
   /(4.0498e7), label = '$R_{T}$' ,color='tab:blue',linewidth=5)
ax.plot( (egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128, (emom)
   /(4.0498e7), label = '$ n_{e}m_{e}u_{ex}(\partial u_{ex}/\partial x)
   +en_eE_x+(\partial n_e T_{ex}/\partial x)  $' ,color='tab:orange',
   linewidth=5)
ax.plot( (egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128, (Rt_lim)
   /(4.0498e7), label = '$-0.71 n_e (\partial T/\partial x) $' ,color='
   tab:green',linewidth=5)
ax.grid(linewidth=2)
```

```
503 ax.set_xlabel('$x/\lambda_D$',fontsize=30)
504 ax.set_xlim(-1,80)
505 ax.legend(loc=1,ncol=1,fontsize=30)
506 ax.tick_params(axis='both', labelsize=30)
507 ax.patch.set_edgecolor('black')
508 ax.patch.set_linewidth('2')
509 plt.tight_layout()
510 # plt.savefig('c{}_emom_LBO.png'.format(case))
511 plt.show()
512
513 # Individual transport terms plot
514
515 # Potential
516 phi = np.zeros(len(Ex))
517 phi =-integ.cumtrapz(Ex,x,initial=0)
518
519 term1 = (Qee+Qei)/(e*eM1x*Ex)
520 term2 = (Qii)/(e*iM1x*Ex)
521 term3 = np.gradient(qx,phi)/(e*eM1x)
522 term4 = np.gradient(iqx,phi)/(e*iM1x)
523 term5 = alpha
524
525 fig, ax = plt.subplots(figsize=(15,10))
526 ax.plot((egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128,term1, label
    = '$(Q_{ee}+Q_{ei}) /e \Gamma_e E_x$' ,color='tab:blue',linewidth=5)
527 ax.plot((egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128,term2, label
    = '$(Q_{ii})/e \Gamma_i E_x $' ,color='tab:green',linewidth=5)
```

```
ax.plot((egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128,term3, label = '$(1/e\Gamma_e) (\partial q_n^e/ \partial \phi_x) $' ,color='tab:orange',linewidth=5)
ax.plot((egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128,term4, label = '$(1/e\Gamma_i) (\partial q_n^i/ \partial \phi_x) $' ,color='tab:red',linewidth=5)
ax.plot((egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128,term5, label = r'$ \alpha $' ,color='tab:purple',linewidth=5)
ax.grid(linewidth=2)
ax.set_xlabel('$x/\lambda_D$',fontsize=30)
ax.set_xlim(-0.25,20)
ax.set_ylim(-12,12)
ax.legend(loc=4,ncol=3,fontsize=30)
ax.tick_params(axis='both',labelsize=30)
ax.patch.set_edgecolor('black')
ax.patch.set_linewidth('2')
plt.tight_layout()
plt.savefig('c{}_effect_LBO.png'.format(case))
plt.show()

# Bohm speed from LBO

dnedx = np.gradient(iM0,x)
dnidx = np.gradient(eM0,x)
reldif = abs(dnedx-dnidx)/abs(dnedx+dnidx)

threshold = 0.1
```

```
se = None

for i, value in enumerate(reldif):
    if value < threshold:
        se = i
        break

A = ((3+(2*alpha))/(e*iM1x[se]))*((np.gradient(iqx,phi))+((Qii)/(Ex)))
B = (alpha/(e*eM1x[se]))*((np.gradient(qx,phi))+((Qee+Qei)/Ex))
C = ((1+alpha)/(e*eM1x[se]))*((np.gradient(qx,phi))+((Qee+Qei)/Ex))

beta = (3-A+B)/(1+C)

uB = np.sqrt(((beta*Tex[se])+(3*Tix[se]))/mi)

data = [
    1.016581833, 1.014726614, 1.009873229, 1.007574911, 1.009898075,
       1.008106354,
    1.005152359, 1.006284263, 1.009376295, 1.010839489, 1.015836433,
       1.019494417,
    1.027680019, 1.032010441, 1.040765939, 1.045285669, 1.05608806,
       1.060946968,
    1.067766002, 1.07092508, 1.082839657, 1.089189365, 1.099486935,
       1.103462404,
    1.101723136, 1.094807476, 1.080189344, 1.074308962, 1.062796665,
       1.07274717,
    1.091348108, 1.101391847, 1.096339688, 1.095925577, 1.095345821,
```

```
        1.09385502,
572     1.090707773, 1.086152548, 1.079372164, 1.06660649, 1.055508305,
        1.057389553,
573     1.063495527, 1.071658649, 1.076988854, 1.069638311, 1.051459807,
        1.037867159,
574     1.04203588, 1.061686847, 1.059483774, 1.050456146, 1.036956114,
        1.054514437,
575     1.057823779, 1.051781302, 1.045965337, 1.024155931, 1.009885061,
        0.99430264,
576     1.004386253, 1.021377243, 1.040670628, 1.053134066, 1.05608806,
        1.05409441,
577     1.040406254, 1.026788228, 1.01099133, 0.999354799, 0.994657593,
        1.002833335,
578     1.018553004, 1.028232857, 1.028591064, 1.035051202, 1.034516939,
        1.019447484,
579     1.003154655, 1.006938954, 1.020802483, 1.02578168, 1.006774624,
        0.990948338,
580     0.979974386, 0.983204455, 0.992232083, 1.005881326, 1.009210651,
        1.005897759,
581     0.998968295, 0.988556845, 0.970118535, 0.963161463, 0.974590938,
        0.979924693
582 ]
583
584 yuzhi = np.array(data)
585 data = [
586     1.016581833, 1.014726614, 1.009873229, 1.007574911, 1.009898075,
        1.008106354,
```

```
    1.005152359, 1.006284263, 1.009376295, 1.010839489, 1.015836433,
       1.019494417,
    1.027680019, 1.032010441, 1.040765939, 1.045285669, 1.05608806,
       1.060946968,
    1.067766002, 1.07092508, 1.082839657, 1.089189365, 1.099486935,
       1.103462404,
    1.101723136, 1.094807476, 1.080189344, 1.074308962, 1.062796665,
       1.07274717,
    1.091348108, 1.101391847, 1.096339688, 1.095925577, 1.095345821,
       1.09385502,
    1.090707773, 1.086152548, 1.079372164, 1.06660649, 1.055508305,
       1.057389553,
    1.063495527, 1.071658649, 1.076988854, 1.069638311, 1.051459807,
       1.037867159,
    1.04203588, 1.061686847, 1.059483774, 1.050456146, 1.036956114,
       1.054514437,
    1.057823779, 1.051781302, 1.045965337, 1.024155931, 1.009885061,
       0.99430264,
    1.004386253, 1.021377243, 1.040670628, 1.053134066, 1.05608806,
       1.05409441,
    1.040406254, 1.026788228, 1.01099133, 0.999354799, 0.994657593,
       1.002833335,
    1.018553004, 1.028232857, 1.028591064, 1.035051202, 1.034516939,
       1.019447484,
    1.003154655, 1.006938954, 1.020802483, 1.02578168, 1.006774624,
       0.990948338,
    0.979974386, 0.983204455, 0.992232083, 1.005881326, 1.009210651,
```

```
       1.005897759,
601    0.998968295, 0.988556845, 0.970118535, 0.963161463, 0.974590938,
       0.979924693
602 ]
603
604 yuzhi = np.array(data)
605
606 Cs = np.sqrt(((Tex)+(3*(Tix)))/mi)
607
608 fig, ax = plt.subplots(figsize=(15,10))
609 ax.plot((egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128,(-iux/Cs),
    label = '$u_i/c_s$' ,color='tab:blue',linewidth=5)
610 ax.plot((egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128,(uB/Cs),
    label = '$u_{Bohm}/c_s$' ,color='tab:blue',linewidth=5,linestyle='--
    ')
611 # ax.plot((egrid[0][0:96]/lamda[case-1])+128,(yuzhi), label = '$
    Prediction (K_n = 50)$' ,color='tab:red',linewidth=3,linestyle='-.')
612 ax.grid(linewidth=2)
613 ax.set_xlabel('$x/\lambda_D$',fontsize=30)
614 ax.set_ylabel('$u_i/C_s$',fontsize=30)
615 ax.set_xlim(-0.25,20)
616 ax.set_ylim(0.7,2.1)
617 ax.legend(loc=1,ncol=1,fontsize=30)
618 ax.tick_params(axis='both', labelsize=30)
619 ax.patch.set_edgecolor('black')
620 ax.patch.set_linewidth('2')
621 plt.tight_layout()
```

```
622 plt.savefig('c{}_bohmspeed_LBO.png'.format(case))
623 plt.show()
```

Appendix E

Momentum, Energy and Bohm speed Computation using BGK (Jupyter notebook)

```
# import libraries
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import matplotlib.gridspec as gridspec
import matplotlib.colors as colors
import matplotlib.patches as patches
import postgkyl as pg
import scipy.integrate as integ
import scipy.optimize as opt
import scipy.special as spc
import scipy.fftpack as fft
from scipy.special import erf, comb, gamma, gammainc
import scipy.constants as const
from scipy.integrate import odeint
```

```
17 import sys
18 import math
19 from glob import glob
20 from mpl_toolkits.axes_grid1 import make_axes_locatable
21
22 # Calculate cell centered values from any edge values 1D array
23 def cell_center_values(cell_edge_values):
24     center = np.array([])
25     for i in range(len(cell_edge_values)-1):
26         avg = (cell_edge_values[i]+cell_edge_values[i+1])/2.0
27         center = np.append(center,avg)
28     return center
29
30 # Initial density and temperatures to calculate debye length and
     thermal velocities
31 n0 = np.array([7e20,4.4e20,3.6e20,8.8e20,4.4e20,2e20,5.7e20,4.4e20,2.8
     e20,1e19,1e19])
32 T0 = np.array([0.5, 0.6,   0.6,   1,     2,     2,   4,     5,     5,
        2.6, 6.0 ])
33 T0 = T0*const.elementary_charge
34
35 lamda = np.zeros(len(n0))
36 vte0 = np.zeros(len(n0))
37 vti0 = np.zeros(len(n0))
38
39 for i in range(len(n0)):
40     lamda[i] = np.sqrt((const.epsilon_0*T0[i])/(n0[i]*const.
```

```
       elementary_charge**2))
    vte0[i] = np.sqrt((T0[i])/const.electron_mass)
    vti0[i] = np.sqrt((T0[i])/const.proton_mass)

#Specify case and time frame
frame = 100
case = 4

#Import gkeyll output files in each case
# Distribution function
data = pg.GData('c{}_elc_{}.bp'.format(case,100))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0),overwrite=True)
egrid = data.getGrid()
edist = data.getValues()[...,0]
data = pg.GData('c{}_ion_{}.bp'.format(case,100))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0),overwrite=True)
igrid = data.getGrid()
idist = data.getValues()[...,0]
# M0
data = pg.GData('c{}_elc_M0_{}.bp'.format(case,100))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate(overwrite=True)
eM0 = data.getValues()[...,0]
data = pg.GData('c{}_ion_M0_{}.bp'.format(case,100))
dg = pg.GInterpModal(data, 2, 'ms')
```

```
dg.interpolate(overwrite=True)
iM0 = data.getValues()[...,0]
# M1i
data = pg.GData('c{}_elc_M1i_{}.bp'.format(case,100))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0,1,2),overwrite=True)
eM1x = data.getValues()[..., 0]
eM1y = data.getValues()[..., 1]
eM1z = data.getValues()[..., 2]
data = pg.GData('c{}_ion_M1i_{}.bp'.format(case,100))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0,1,2),overwrite=True)
iM1x = data.getValues()[..., 0]
iM1y = data.getValues()[..., 1]
iM1z = data.getValues()[..., 2]
# M2ij
data = pg.GData('c{}_elc_M2ij_{}.bp'.format(case,100))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0,1,2,3,4,5),overwrite=True)
eM2xx = data.getValues()[..., 0]
eM2yy = data.getValues()[..., 3]
eM2zz = data.getValues()[..., 5]
data = pg.GData('c{}_ion_M2ij_{}.bp'.format(case,100))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0,1,2,3,4,5),overwrite=True)
iM2xx = data.getValues()[..., 0]
iM2yy = data.getValues()[..., 3]
```

```
iM2zz = data.getValues()[..., 5]
# Udrift
data = pg.GData('c{}_elc_Udrift_{}.bp'.format(case,100))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0,1,2),overwrite=True)
eUdrift = data.getValues()
eux = data.getValues()[..., 0]
euy = data.getValues()[..., 1]
euz = data.getValues()[..., 2]
data = pg.GData('c{}_ion_Udrift_{}.bp'.format(case,100))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0,1,2),overwrite=True)
iUdrift = data.getValues()
iux = data.getValues()[..., 0]
iuy = data.getValues()[..., 1]
iuz = data.getValues()[..., 2]
# Field
data = pg.GData('c{}_field_{}.bp'.format(case,frame))
dg = pg.GInterpModal(data, 2, 'ms')
dg.interpolate((0),overwrite=True)
Ex = data.getValues()[...,0]

#since cases are run with different resolution
if case == 7:
    xres = 72
    yres = 48
    zres = 48
```

```
121 elif case == 10:
122     xres = 54
123     yres = 36
124     zres = 36
125 elif case == 9:
126     xres = 54
127     yres = 36
128     zres = 36
129 elif case == 5:
130     xres = 54
131     yres = 36
132     zres = 36
133 else:
134     xres = 36
135     yres = 24
136     zres = 24
137 
138 # Calculate electron and ion heat flux
139 eux = eM1x/eM0
140 evx = cell_center_values(egrid[1])
141 eveldiff = np.zeros((384,xres))
142 for i in range(384):
143     eveldiff[i] = evx-eux[i]
144 eveldiff3 = eveldiff**3
145 efvdiff = edist.copy()
146 for i in range(384):
147     for j in range(xres):
```

```
        efvdiff[i,j,:] = eveldiff3[i,j]*efvdiff[i,j,:]
# Integration
edvx = egrid[1][1]-egrid[1][0]
edvy = egrid[2][1]-egrid[2][0]
edvz = egrid[3][1]-egrid[3][0]
Iz = np.trapz((efvdiff*edvz),axis=3)
Iy = np.trapz((Iz*edvy),axis=2)
I = np.trapz((Iy*edvx),axis=1)
qx = const.electron_mass*I

iux = iM1x/iM0
ivx = cell_center_values(igrid[1])
iveldiff = np.zeros((384,xres))
for i in range(384):
    iveldiff[i] = ivx-iux[i]
iveldiff3 = iveldiff**3
ifvdiff = idist.copy()
for i in range(384):
    for j in range(xres):
        ifvdiff[i,j,:] = iveldiff3[i,j]*ifvdiff[i,j,:]
# Integration
idvx = igrid[1][1]-igrid[1][0]
idvy = igrid[2][1]-igrid[2][0]
idvz = igrid[3][1]-igrid[3][0]
iIz = np.trapz((ifvdiff*idvz),axis=3)
iIy = np.trapz((iIz*idvy),axis=2)
iI =  np.trapz((iIy*idvx),axis=1)
```

```
175 iqx = const.proton_mass*iI
176
177 # calculate collision integral and isotropization from LBO operator
178 me = const.electron_mass
179 mi = const.proton_mass
180 pi = math.pi
181 e = const.elementary_charge
182 ep0 = const.epsilon_0
183
184 # Temperatures
185 Tex = me*((eM2xx/eM0)-(eux)**2)
186 Tey = me*((eM2yy/eM0)-(euy)**2)
187 Tez = me*((eM2zz/eM0)-(euz)**2)
188
189 Tix = mi*((iM2xx/iM0)-(iux)**2)
190 Tiy = mi*((iM2yy/iM0)-(iuy)**2)
191 Tiz = mi*((iM2zz/iM0)-(iuz)**2)
192
193 Vt2_e = (Tex+Tey+Tez)/(3*me)
194 Vt2_i = (Tix+Tiy+Tiz)/(3*mi)
195
196 #anisotropic Collision frequencies
197 nu_ei = (np.sqrt(pi)*iM0*(e**4)*13)/(np.sqrt(2)*((4*pi*ep0)**2)*np.sqrt
      (me*Tex)*Tey)
198 nu_ee = (eM0*nu_ei)/(iM0*np.sqrt(2))
199 nu_ie = nu_ee*(me/mi)*(eM0/iM0)
200 nu_ii = (nu_ee)*(iM0/iM0)*(Tey/Tiy)*(np.sqrt(me*Tex)/np.sqrt(mi*Tix))
```

```
201
202 #BGK terms
203 delta_ei = (2*me*eM0*nu_ei)/((me*eM0*nu_ei)+(mi*iM0*nu_ie)) #ok
204 alpha_ei = (me*eM0*nu_ei*delta_ei)/(me+mi)
205
206 Udiff = eUdrift-iUdrift
207 Udiff_dot = np.einsum('ij,ij->i',Udiff,Udiff) #ok
208
209 Vt2_ei = (Vt2_e) - ((alpha_ei/(3*me*eM0*nu_ei))*((3*((me*Vt2_e)-(mi*
    Vt2_i)))-(mi*Udiff_dot)+((4)*(alpha_ei/(me*eM0*nu_ei))*((me+mi)**2)
    *(Udiff_dot)))) #ok
210 U_ei = eUdrift - (((alpha_ei[:, np.newaxis]*(me+mi))/(2*me*eM0[:, np.
    newaxis]*nu_ei[:, np.newaxis]))*(Udiff)) #ok
211
212 #
213
214 delta_ee = (2*me*eM0*nu_ee)/((me*eM0*nu_ee)+(me*eM0*nu_ee)) #ok
215 alpha_ee = (me*eM0*nu_ee*delta_ee)/(me+me)
216
217 Udiff = eUdrift-eUdrift
218 Udiff_dot = np.einsum('ij,ij->i',Udiff,Udiff) #ok
219
220 Vt2_ee = (Vt2_e) - ((alpha_ee/(3*me*eM0*nu_ee))*((3*((me*Vt2_e)-(me*
    Vt2_e)))-(me*Udiff_dot)+((4)*(alpha_ee/(me*eM0*nu_ee))*((me+me)**2)
    *(Udiff_dot)))) #ok
221 U_ee = eUdrift - (((alpha_ee[:, np.newaxis]*(me+me))/(2*me*eM0[:, np.
    newaxis]*nu_ee[:, np.newaxis]))*(Udiff)) #ok
```

```
222
223 #
224
225 delta_ii = (2*mi*iM0*nu_ii)/((mi*iM0*nu_ii)+(mi*iM0*nu_ii)) #ok
226 alpha_ii = (mi*iM0*nu_ii*delta_ii)/(mi+mi)
227
228 Udiff = iUdrift-iUdrift
229 Udiff_dot = np.einsum('ij,ij->i',Udiff,Udiff) #ok
230
231 Vt2_ii = (Vt2_i) - ((alpha_ii/(3*mi*iM0*nu_ii))*((3*((mi*Vt2_i)-(mi*
    Vt2_i)))-(mi*Udiff_dot)+((4)*(alpha_ii/(mi*iM0*nu_ii))*((mi+mi)**2)
    *(Udiff_dot)))) #ok
232 U_ii = iUdrift - (((alpha_ii[:, np.newaxis]*(mi+mi))/(2*mi*iM0[:, np.
    newaxis]*nu_ii[:, np.newaxis]))*(Udiff)) #ok
233
234 # Compute Qei adn Qee
235
236 f_Mei = np.zeros_like(edist)
237 f_Mee = np.zeros_like(edist)
238
239 for i in range(384):
240     for j in range(xres):
241         for k in range(yres):
242             for l in range(zres):
243                 f_Mei[i,j,k,l]= (eM0[i]/((2*math.pi*Vt2_ei[i])**(3/2)))
                    *(math.exp(-((vx[j]-U_ei[i][0])**2+(vy[k]-U_ei[i
                    ][1])**2+(vz[l]-U_ei[i][2])**2)/(2*Vt2_ei[i])))
```

```
                f_Mee[i,j,k,l]= (eM0[i]/((2*math.pi*Vt2_ee[i])**(3/2)))
                   *(math.exp(-((vx[j]-U_ee[i][0])**2+(vy[k]-U_ee[i
                   ][1])**2+(vz[l]-U_ee[i][2])**2)/(2*Vt2_ee[i])))

dfeidt = np.zeros_like(edist)
fMeifd = f_Mei-edist

dfeedt = np.zeros_like(edist)
fMeefd = f_Mee-edist

for i in range(384):
    dfeidt[i,:] = nu_ei[i]*fMeifd[i,:]
    dfeedt[i,:] = nu_ee[i]*fMeefd[i,:]

vxmux = np.zeros((384,xres))

for i in range(384):
    vxmux[i] = vx-eux[i]

vxmux2 =  vxmux**2

vxmuxdfeidt = np.zeros_like(edist)
vxmuxdfeedt = np.zeros_like(edist)

for i in range(384):
    for j in range(xres):
        vxmuxdfeidt[i,j,:] = vxmux2[i,j]*dfeidt[i,j,:]
```

```
269         vxmuxdfeedt[i,j,:] = vxmux2[i,j]*dfeedt[i,j,:]
270
271 dvx = vx[1]-vx[0]
272 dvy = vy[1]-vy[0]
273 dvz = vz[1]-vz[0]
274
275 Izei = np.trapz((vxmuxdfeidt*dvz),axis=3) # --NEW--
276 Iyei = np.trapz((Izei*dvy),axis=2)
277 Ixei = np.trapz((Iyei*dvx),axis=1)
278 Qei = Ixei*me
279
280 Izee = np.trapz((vxmuxdfeedt*dvz),axis=3) # --NEW--
281 Iyee = np.trapz((Izee*dvy),axis=2)
282 Ixee = np.trapz((Iyee*dvx),axis=1)
283 Qee  = Ixee*me
284
285 #Compute Qii
286
287 vx = cell_center_values(igrid[1])
288 vy = cell_center_values(igrid[2])
289 vz = cell_center_values(igrid[3])
290
291 f_Mii = np.zeros_like(idist)
292
293 for i in range(384):
294     for j in range(xres):
295         for k in range(yres):
```

```
296             for l in range(zres):
297                 f_Mii[i,j,k,l]= (iM0[i]/((2*math.pi*Vt2_ii[i])**(3/2)))
                       *(math.exp(-((vx[j]-U_ii[i][0])**2+(vy[k]-U_ii[i
                       ][1])**2+(vz[l]-U_ii[i][2])**2)/(2*Vt2_ii[i])))
298
299 dfiidt = np.zeros_like(idist)
300 fMiifd = f_Mii-idist
301
302 for i in range(384):
303     dfiidt[i,:] = nu_ii[i]*fMiifd[i,:]
304
305 vxmux = np.zeros((384,xres))
306 for i in range(384):
307     vxmux[i] = vx-iux[i]
308
309 vxmux2 = vxmux**2
310
311 vxmuxdfiidt = np.zeros_like(idist)
312
313 for i in range(384):
314     for j in range(xres):
315         vxmuxdfiidt[i,j,:] = vxmux2[i,j]*dfiidt[i,j,:]
316
317 dvx = vx[1]-vx[0]
318 dvy = vy[1]-vy[0]
319 dvz = vz[1]-vz[0]
320
```

```
Izii = np.trapz((vxmuxdfiidt*dvz),axis=3) # --NEW--
Iyii = np.trapz((Izii*dvy),axis=2)
Ixii = np.trapz((Iyii*dvx),axis=1)
Qii = Ixii*me

# Energy plot

x = cell_center_values(egrid[0])
dTexdx = np.gradient(Tex,x)
deuxdx = np.gradient(eux,x)

EnergyLHS = (eM0*eux*dTexdx) + (2*eM0*Tex*deuxdx)

dqedx = -1*(np.gradient(qx,x))

EnergyISO = (EnergyLHS) - (Qee) - (Qei)

fig, ax = plt.subplots(figsize=(15,10))
ax.plot((egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128, ((dqedx)
   /(1.6982e13)), label = '$-(\partial q_n^e/\partial x)$' ,color='tab:
   blue',linewidth=5)
ax.plot((egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128, ((EnergyISO)
   /(1.6982e13)), label = '$ n_e u_{ex} (\partial T_{ex}/\partial x)+2
   n_eT_{ex}(\partial u_{ex}/\partial x) - Q_{ee} -Q_{ei} $' ,color='
   tab:orange',linewidth=5)
ax.plot((egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128, ((EnergyLHS)
   /1.6982e13), label = '$ n_e u_{ex} (\partial T_{ex}/\partial x)+2
```

```
   n_eT_{ex}(\partial u_{ex}/\partial x) $' ,color='tab:green',
   linewidth=5)
ax.grid(linewidth=2)
ax.set_xlabel('$x/\lambda_D$',fontsize=30)
ax.set_xlim(-1,80)
ax.set_ylim(-0.025,0.175)
ax.legend(loc=1,ncol=1,fontsize=30)
ax.tick_params(axis='both', labelsize=30)
ax.patch.set_edgecolor('black')
ax.patch.set_linewidth('2')
plt.tight_layout()
plt.savefig('c{}_eenergy_LBO.png'.format(case))
plt.show()

# Thermal force

vdfdt = np.zeros_like(edist)

for i in range(384):
    for j in range(xres):
        vdfdt[i,j,:] = vx[j]*dfeidt[i,j,:]

dvx = vx[1]-vx[0]
dvy = vy[1]-vy[0]
dvz = vz[1]-vz[0]
Iz = np.trapz((vdfdt*dvz),axis=3) # --NEW--
Iy = np.trapz((Iz*dvy),axis=2)
```

```
I  = np.trapz((Iy*dvx),axis=1)
Rt = I*me

alpha = -Rt/(eM0*dTexdx)

#Momentum plot

Pex = eM0*Tex
dpexdx = np.gradient(Pex,x)

emom = (eM0*me*eux*deuxdx)+(dpexdx)+(e*eM0*Ex)
Rt_lim = (-1*0.71*eM0*dTexdx)

fig, ax = plt.subplots(figsize=(15,10))
ax.plot( (egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128, (Rt)
   /(4.0498e7), label = '$R_{T}$' ,color='tab:blue',linewidth=5)
ax.plot( (egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128, (emom)
   /(4.0498e7), label = '$ n_{e}m_{e}u_{ex}(\partial u_{ex}/\partial x)
   +en_eE_x+(\partial n_e T_{ex}/\partial x)  $' ,color='tab:orange',
   linewidth=5)
ax.plot( (egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128, (Rt_lim)
   /(4.0498e7), label = '$-0.71 n_e (\partial T/\partial x) $' ,color='
   tab:green',linewidth=5)
ax.grid(linewidth=2)
ax.set_xlabel('$x/\lambda_D$',fontsize=30)
ax.set_xlim(-1,80)
ax.legend(loc=1,ncol=1,fontsize=30)
```

```
ax.tick_params(axis='both', labelsize=30)
ax.patch.set_edgecolor('black')
ax.patch.set_linewidth('2')
plt.tight_layout()
# plt.savefig('c{}_emom_LBO.png'.format(case))
plt.show()

# Individual transport terms plot

# Potential
phi = np.zeros(len(Ex))
phi =-integ.cumtrapz(Ex,x,initial=0)

term1 = (Qee+Qei)/(e*eM1x*Ex)
term2 = (Qii)/(e*iM1x*Ex)
term3 = np.gradient(qx,phi)/(e*eM1x)
term4 = np.gradient(iqx,phi)/(e*iM1x)
term5 = alpha

fig, ax = plt.subplots(figsize=(15,10))
ax.plot((egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128,term1, label
   = '$(Q_{ee}+Q_{ei}) /e \Gamma_e E_x$' ,color='tab:blue',linewidth=5)
ax.plot((egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128,term2, label
   = '$(Q_{ii})/e \Gamma_i E_x $' ,color='tab:green',linewidth=5)
ax.plot((egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128,term3, label
   = '$(1/e\Gamma_e) (\partial q_n^e/ \partial \phi_x) $' ,color='tab:
   orange',linewidth=5)
```

```
411 ax.plot((egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128,term4, label
    = '$(1/e\Gamma_i) (\partial q_n^i/ \partial \phi_x) $' ,color='tab:
    red',linewidth=5)
412 ax.plot((egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128,term5, label
    = r'$ \alpha $' ,color='tab:purple',linewidth=5)
413 ax.grid(linewidth=2)
414 ax.set_xlabel('$x/\lambda_D$',fontsize=30)
415 ax.set_xlim(-0.25,20)
416 ax.set_ylim(-12,12)
417 ax.legend(loc=4,ncol=3,fontsize=30)
418 ax.tick_params(axis='both',labelsize=30)
419 ax.patch.set_edgecolor('black')
420 ax.patch.set_linewidth('2')
421 plt.tight_layout()
422 plt.savefig('c{}_effect_LBO.png'.format(case))
423 plt.show()
424
425 # Bohm speed from LBO
426
427 dnedx = np.gradient(iM0,x)
428 dnidx = np.gradient(eM0,x)
429 reldif = abs(dnedx-dnidx)/abs(dnedx+dnidx)
430
431 threshold = 0.1
432 se = None
433
434 for i, value in enumerate(reldif):
```

```
435     if value < threshold:
436         se = i
437         break
438 
439 A = ((3+(2*alpha))/(e*iM1x[se]))*((np.gradient(iqx,phi))+((Qii)/(Ex)))
440 B = (alpha/(e*eM1x[se]))*((np.gradient(qx,phi))+((Qee+Qei)/Ex))
441 C = ((1+alpha)/(e*eM1x[se]))*((np.gradient(qx,phi))+((Qee+Qei)/Ex))
442 
443 beta = (3-A+B)/(1+C)
444 
445 uB = np.sqrt(((beta*Tex[se])+(3*Tix[se]))/mi)
446 
447 data = [
448     1.016581833, 1.014726614, 1.009873229, 1.007574911, 1.009898075,
        1.008106354,
449     1.005152359, 1.006284263, 1.009376295, 1.010839489, 1.015836433,
        1.019494417,
450     1.027680019, 1.032010441, 1.040765939, 1.045285669, 1.05608806,
        1.060946968,
451     1.067766002, 1.07092508, 1.082839657, 1.089189365, 1.099486935,
        1.103462404,
452     1.101723136, 1.094807476, 1.080189344, 1.074308962, 1.062796665,
        1.07274717,
453     1.091348108, 1.101391847, 1.096339688, 1.095925577, 1.095345821,
        1.09385502,
454     1.090707773, 1.086152548, 1.079372164, 1.06660649, 1.055508305,
        1.057389553,
```

```
    1.063495527, 1.071658649, 1.076988854, 1.069638311, 1.051459807,
        1.037867159,
    1.04203588, 1.061686847, 1.059483774, 1.050456146, 1.036956114,
        1.054514437,
    1.057823779, 1.051781302, 1.045965337, 1.024155931, 1.009885061,
        0.99430264,
    1.004386253, 1.021377243, 1.040670628, 1.053134066, 1.05608806,
        1.05409441,
    1.040406254, 1.026788228, 1.01099133, 0.999354799, 0.994657593,
        1.002833335,
    1.018553004, 1.028232857, 1.028591064, 1.035051202, 1.034516939,
        1.019447484,
    1.003154655, 1.006938954, 1.020802483, 1.02578168, 1.006774624,
        0.990948338,
    0.979974386, 0.983204455, 0.992232083, 1.005881326, 1.009210651,
        1.005897759,
    0.998968295, 0.988556845, 0.970118535, 0.963161463, 0.974590938,
        0.979924693
]

yuzhi = np.array(data)
data = [
    1.016581833, 1.014726614, 1.009873229, 1.007574911, 1.009898075,
        1.008106354,
    1.005152359, 1.006284263, 1.009376295, 1.010839489, 1.015836433,
        1.019494417,
    1.027680019, 1.032010441, 1.040765939, 1.045285669, 1.05608806,
```

```
      1.060946968,
471   1.067766002, 1.07092508, 1.082839657, 1.089189365, 1.099486935,
      1.103462404,
472   1.101723136, 1.094807476, 1.080189344, 1.074308962, 1.062796665,
      1.07274717,
473   1.091348108, 1.101391847, 1.096339688, 1.095925577, 1.095345821,
      1.09385502,
474   1.090707773, 1.086152548, 1.079372164, 1.06660649, 1.055508305,
      1.057389553,
475   1.063495527, 1.071658649, 1.076988854, 1.069638311, 1.051459807,
      1.037867159,
476   1.04203588, 1.061686847, 1.059483774, 1.050456146, 1.036956114,
      1.054514437,
477   1.057823779, 1.051781302, 1.045965337, 1.024155931, 1.009885061,
      0.99430264,
478   1.004386253, 1.021377243, 1.040670628, 1.053134066, 1.05608806,
      1.05409441,
479   1.040406254, 1.026788228, 1.01099133, 0.999354799, 0.994657593,
      1.002833335,
480   1.018553004, 1.028232857, 1.028591064, 1.035051202, 1.034516939,
      1.019447484,
481   1.003154655, 1.006938954, 1.020802483, 1.02578168, 1.006774624,
      0.990948338,
482   0.979974386, 0.983204455, 0.992232083, 1.005881326, 1.009210651,
      1.005897759,
483   0.998968295, 0.988556845, 0.970118535, 0.963161463, 0.974590938,
      0.979924693
```

```
]

yuzhi = np.array(data)

Cs = np.sqrt(((Tex)+(3*(Tix)))/mi)

fig, ax = plt.subplots(figsize=(15,10))
ax.plot((egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128,(-iux/Cs),
    label = '$u_i/c_s$' ,color='tab:blue',linewidth=5)
ax.plot((egrid[0][0:(len(egrid[0])-1)]/lamda[case-1])+128,(uB/Cs),
    label = '$u_{Bohm}/c_s$' ,color='tab:blue',linewidth=5,linestyle='--
    ')
# ax.plot((egrid[0][0:96]/lamda[case-1])+128,(yuzhi), label = '$
    Prediction (K_n = 50)$' ,color='tab:red',linewidth=3,linestyle='-.')
ax.grid(linewidth=2)
ax.set_xlabel('$x/\lambda_D$',fontsize=30)
ax.set_ylabel('$u_i/C_s$',fontsize=30)
ax.set_xlim(-0.25,20)
ax.set_ylim(0.7,2.1)
ax.legend(loc=1,ncol=1,fontsize=30)
ax.tick_params(axis='both', labelsize=30)
ax.patch.set_edgecolor('black')
ax.patch.set_linewidth('2')
plt.tight_layout()
plt.savefig('c{}_bohmspeed_LBO.png'.format(case))
plt.show()
```