BACHELOR

Mapping a plasma sheath with dust particles

Meekes, Wouter B.

Award date:
2019

Mapping a Plasma Sheath with Dust Particles

W.B. Meekes

Eindhoven University of Technology
Department of Applied Physics
Elementary Processes in Gas Discharges

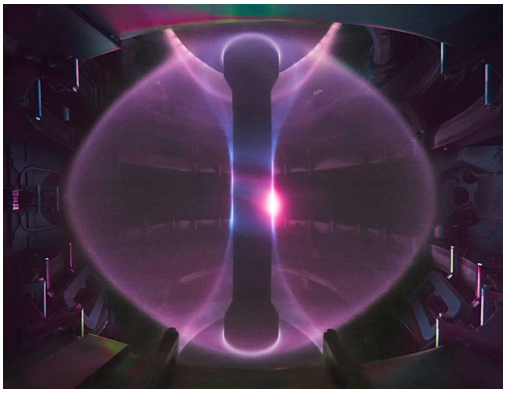
Abstract

A 1D analytical model for the forces acting on a dust particle in the sheath of an argon plasma between two parallel-plate RF electrodes has been designed. Particular attention has been paid to versatility and user-friendliness of the model, to make it possible to test a wide variety of options for the parameters present in the model or apply the model to a different setup. The model was verified using data from a centrifuge setup, where the force of gravity was varied to find equilibrium positions for the particle. At each of these positions, the net force should be zero. The combinations of parameters that resulted in the net force function most equal to zero would be the best model. Calculations using the model demonstrate a proof of concept; the code works, but the parameters should be investigated further.

23rd June 2019
Eindhoven University of Technology

Contents

(a) Lightning discharge.[2]



(b) Plasma in the ITER fusion reactor.[3]

(c) Plasma discharge on a cut grape in a microwave.[4]

(d) The sun is also a plasma. [5]

Figure 1: Examples of plasmas that can be seen on or from earth.

1 Introduction

If you ask people how many states of matter there are, most will respond that there are 3: solid, liquid and gas. In a solid, the molecules have a fixed position relative to each other. Heat a solid to a high enough temperature, and the thermal energy of the molecules will overcome the binding energy between them. Although they still attract each other and are closely packed, the molecules can move with respect to one another. The solid has melted and is now a liquid. Upon heating the liquid even further, the molecules will start vibrating more and more until they move so fast that their mutual attraction cannot keep them together anymore. The liquid vaporises into the gas phase. These three states are, however, not the only states of matter, nor are they the most abundant. Of all visible matter in the universe, 99% is estimated to be in the plasma phase [1]. A plasma is essentially an ionized gas. Using even more heat or strong electric fields on a gas, electrons can be ripped from their respective molecules to form the plasma. On earth, plasmas can naturally be found in the form of auroras and lightning, and they are used in e.g. lamps, the semiconductor industry or fusion reactors. A fun experiment to make plasmas at home is placing a grape or lit match in a microwave (outside, for the sake of safety). However, the more significant contributions to the plasma content in the universe come from space. Stars and nebulae are completely in the plasma phase. Figure 1 shows a few examples of plasmas.

A plasma has a few very interesting and unique properties. It behaves like a gas and thus reacts to mechanical forces in the same way. However, because the gas contains free charge carriers, it can be influenced using electric or magnetic fields as well. Furthermore, the source that supplies energy to the plasma does not only

ionize the molecules, but also excites the electrons in the plasma. When these excited electrons fall back to their ground states, they release a photon, causing the plasma to emit light.

Anyone who ever forgot to clean for more than a week will know that dust finds a way into every nook and cranny. However, although dust is a nuisance in a household, it can be outright problematic in other settings. For instance, in fusion reactors, wall material can come off the wall and be confined inside the electric and magnetic fields, where it agglomerates to form nanometre to micrometre-sized particles. These particles negatively affect the fusion plasma. [6] A plasma that contains such particles is called a dusty plasma. In the semiconductor industry, dust particles can damage the microelectronics, but in solar cells they have been shown to be capable of increasing the efficiency and durability of the devices. i Cabarrocas et al. [7], for instance, showed that dust particles can help to form a more homogeneous silicon layer in solar cells which can improve the cells' electric properties.

Both the advantages and disadvantages of dust sparked interest in dusty plasmas. The advantage of dusty plasmas that is exploited in this thesis is that of diagnosis. Dust particles floating freely in a plasma are influenced by the plasma much more than for instance a Langmuir probe, and can thus provide information about the plasma in a much less invasive manner. Which leads to the goal of this thesis. Using data from a particular plasma setup, a 1-dimensional model will be made for the forces that act on a dust particle as a function of the position of the particle at the edge of the plasma.

Several expressions will be determined for the five most significant forces, as well as the parameters they depend on. From the real-life setup, equilibrium positions for the dust particle in the sheath have been found, at which the net force on the particle should thus be equal to zero. Using matlab, this net force will be determined for various combinations of parameters, to find the combination in which it is most equal to zero.

This thesis will begin by explaining the theory necessary to be able to quantify the plasma parameters in section 2. In section 3, the scripts making up the model will be explained, as well as how to use them. The results of running the scripts with the current set of parameters are shown in section 4 and a brief conclusion will be drawn based on those results in section 5. Section 6 will take a critical look back at what could be improved or added.

Table 1: Overview of all quantities.

Sym	Code	Description	Unit	Sym	Code	Description	Unit
k_B	kb	Boltzmann's constant	J/K	e	el	Electron charge	C
ϵ_0	eps0	Vacuum permittivity	C/Vm	m_{amu}	kovamu	Atomic mass unit mass	kg
T_b	Tb	Neutral bulk temperature	K	T_{el}	Tel	Electrode temperature	K
$T_{e,b}$	Teb	Electron bulk temperature	K	T_n	Tn	Neutral atom temperature	K
T_i	Ti	Ion temperature	K	T_e	Te	Electron temperature	K
$T_{i,a}$	TiA	Average T_i	K	$T_{e,a}$	TeA	Average T_e	K
∇T	gradT	Temperature gradient	K/m	s	s	Sheath thickness	m
s_0	s0	Pre-sheath thickness	m	$\lambda_{mf,i}$	lmfi	Ion mean free path	m
$\lambda_{mf,e}$	lmfe	Electron mean free path	m	σ	sig	scattering cross section	m^2
$\sigma_{i,n}$	sigin	Ion-neutral σ	m^2	$\sigma_{e,n}$	sigen	Electron-neutral σ	m^2
$\lambda_{D,i}$	lDi	Ion Debye length	m	$\lambda_{D,e}$	lDe	Electron Debye length	m
λ_D	lD	Total Debye length	m	m_{in}	min	Inertial mass	kg
m_g	mg	Gravitational mass	kg	m_i	mi	Ion mass	kg
m_e	me	Electron mass	kg	m_n	mn	Neutral atom mass	kg
$\nu_{p,i}$	ompi	Ion plasma frequency	Hz	$\nu_{p,e}$	ompe	Electron plasma frequency	Hz
p_i	pi	Ion momentum	kg m/s	$\overrightarrow{p}$	dip	Particle dipole moment	Cm
r_p	rp	Particle radius	m	ρ_p	rhop	Particle density	kg/m^3
Q_p	Qp	Particle charge	C	V_p	Vp	Particle potential	V
Φ	Phi	Sheath potential	V	V_0	V0	Sheath edge potential	V
ΔV	DeltaV	Electrode potential	V	V_{rf}	Vrf	Applied voltage	V
ν_{rf}	nurf	Driver voltage frequency	Hz	$\overrightarrow{E}$	E	Electric field	V/m
p	p	pressure	Pa	n_g	ng	Gas density	$1/m^3$
$n_{i,0}$	ni0	Bulk ion density	$1/m^3$	$n_{e,0}$	ne0	Bulk electron density	$1/m^3$
n_i	nis	Ion density	$1/m^3$	n_e	nes	Electron density	$1/m^3$
$u_{i,B}$	uibohm	Ion sheath edge velocity	m/s	u_n	un	Neutral drift velocity	m/s
u_i	ui	Ion velocity	m/s	$u_{i,ef}$	uieff	Effective ion velocity	m/s
u_p	up	Particle velocity	m/s	v_{T_n}	vTn	Neutral thermal velocity	m/s
v_{T_i}	vTi	Ion thermal velocity	m/s	v_{T_e}	vTe	Electron thermal velocity	m/s
k_T	kT	Thermal conductivity	W/mK	ρ_c	rhoc	Charge density	C m^{-3}
σ_c	sigmac	Surface charge density	C m^{-3}	b	b	Impact parameter	m
b_{coll}	bcoll	Collection b	m	b_0	b0	Asymptotic b	m
Δ	LOG	Coulomb logarithm	-	ω	omega	Angular frequency	rad/s
$\overrightarrow{r}$	r	Centrifuge arm length	m	$\overrightarrow{F_g*}$	Fg	Perceived gravity	N
$\overrightarrow{F_i}$	Fi	Ion drag force	N	$\overrightarrow{F_N}$	FN	Neutral drag force	N
$\overrightarrow{F_{Th}}$	FTh	Thermophoretic force	N	$\overrightarrow{F_C}$	FC	Coulomb force	N
$\overrightarrow{F_{C,mon}}$	FCmon	Coulomb monopole term	N	$\overrightarrow{F_{C,dip}}$	FCdip	Coulomb dipole term	N
$\overrightarrow{F_S}$	FS	Sum force	N	$\overrightarrow{F_{net}}$	Fnet	Net force ($F_S + F_g$)	N
$\overrightarrow{F_r}$	Fr	Centripetal force	N	$\overrightarrow{F_{grav}}$	Fgrav	Real gravity	N

2 Theoretical background

This section describes all theory necessary to understand the expressions that have been tried for the model. For all expressions with multiple options, a most likely option will be given with a brief rationale. First, the definitions of the variables used will be given in subsection 2.1. Next the setup will be described in subsection 2.2. Then different expressions for important plasma parameters will be derived in subsection 2.3. Finally, expressions for the forces considered will be discussed in subsection 2.4.

2.1 Definitions

In plasma physics, different authors often use different symbols and different units for the same quantities. To avoid confusion, this chapter will thus start with an overview of all quantities used, listing their symbol - both the report and code version - their meaning and their units. This overview is in table 1.

2.2 Setup

Although the development of the model was done using nothing but a laptop (HP ZBook Studio), the internet and a few books, the model is based on a real-life setup. This subsection will briefly describe the real setup and outline how the boundary conditions were chosen with respect to the setup. In other words, it explains how the model ties in with the real world.

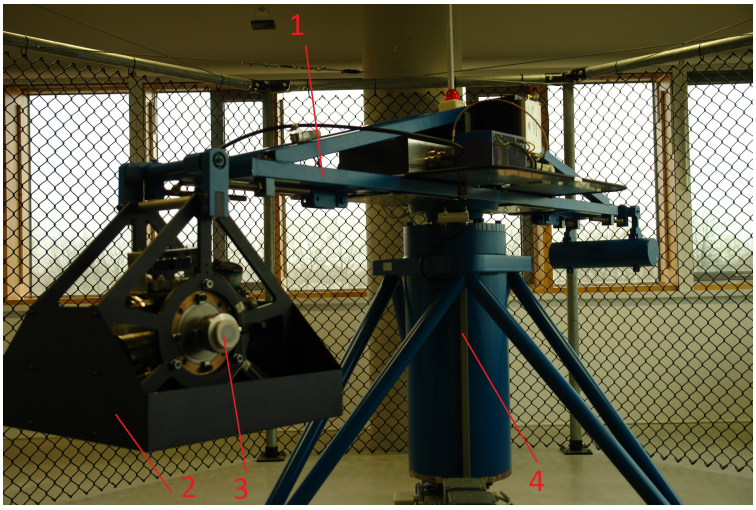
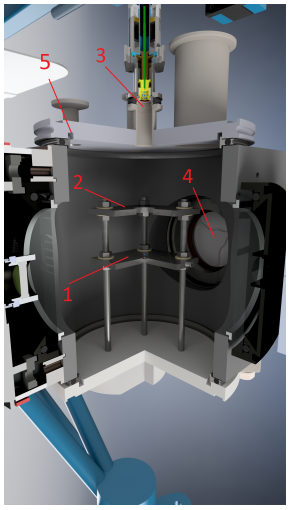


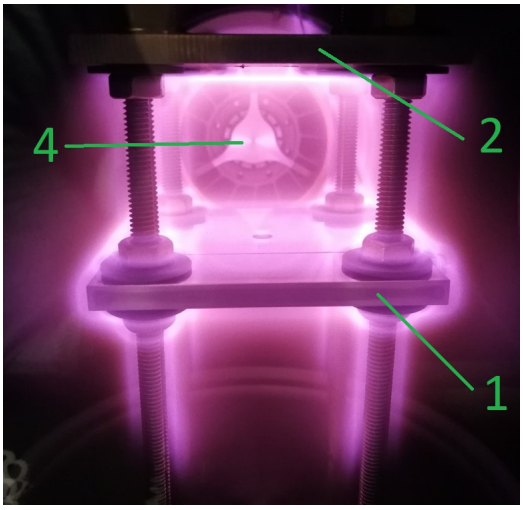
Figure 2: Image of the centrifuge, with attached to the arm (1) the gondola (2) and the plasma chamber (3). (4) indicates the axis of rotation.

The setup (in baseline) consists of a freely swinging gondola attached to one arm of a centrifuge. The plasma is produced in a vacuum chamber in the gondola, shown in figure 2. This vacuum chamber - shown in more detail in figure 3 - contains two parallel electrodes, separated by 4 cm and oriented parallel to the ground (when the centrifuge is stationary). The upper electrode is grounded, the lower has an 80 V, 13.56 MHz RF voltage V_{rf} over it.

The reason the plasma is in a swinging gondola on a centrifuge is so that gravity can be used as a variable. As the centrifuge spins, the gondola, the plasma and the particles in the plasma will appear to experience an inertial force with magnitude $\overrightarrow{F_r} = m_{in} \cdot \omega^2 \cdot \overrightarrow{r}$, where m_{in} is an object's inertial mass, and ω and $\overrightarrow{r}$ are the angular frequency and radius of rotation, respectively [8]. $\overrightarrow{F_r}$ points away from the axis of rotation of the centrifuge. Bear in mind that it is not an actual force. By Newton's first law, the gondola simply does not want to change its direction. The reason the gondola has a tendency to swing outward is that its trajectory at any given point in time points away from the circle that the centrifuge is making it follow, as demonstrated in figure 4.



(a) Schematic image of the inside of the plasma chamber without plasma.



(b) Photograph of the inside of the plasma chamber with plasma.

Figure 3: Images of the plasma chamber. 1: RF electrode. 2: Grounded electrode. 3: Particle dispenser. 4: Gas outlet. 5: Gas inlet.

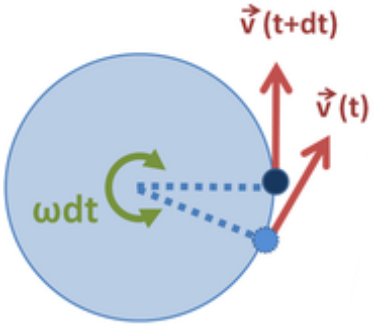


Figure 4: The velocity of an object following a circular trajectory points away from the trajectory. It is made to change direction constantly, which is the cause of the inertial force.

The other force important for the workings of this setup is the force of gravity. At sea level, it can be taken to have a magnitude of $\overrightarrow{F_{grav}} = m_g \cdot \overrightarrow{g}$, where m_g is the object's gravitational mass, and $\overrightarrow{g}$ the gravitational acceleration. The total force experienced is then $\overrightarrow{F_{tot}} = m_g \cdot \overrightarrow{g} + m_{in} \cdot \omega^2 \cdot \overrightarrow{r}$. However, by Einstein's equivalence principle, $m_{in} = m_g = m$ [9], so that $\overrightarrow{F_{tot}} = m \cdot (\overrightarrow{g} + \omega^2 \cdot \overrightarrow{r})$. Since m is a scalar, the direction of F_{tot} is determined entirely by the terms in brackets, and is thus independent of mass. Because the gondola will orient itself along F_{tot}, and the force on particles in the plasma in the gondola will also be parallel to F_{tot}, this total force will still point straight towards the lower electrode, as regarded from the reference frame of the particle. This is shown in figure 5. The difference is that the combined inertial and gravitational forces are greater than the gravitational force alone. This new force is called the 'perceived gravity', $F_{g^*} = m_p g^*$, where $g^* = \sqrt{g^2 + \omega^4 r^2}$. Transforming g into $g*$ opens gravity up as a known variable for experiments, by increasing ω.

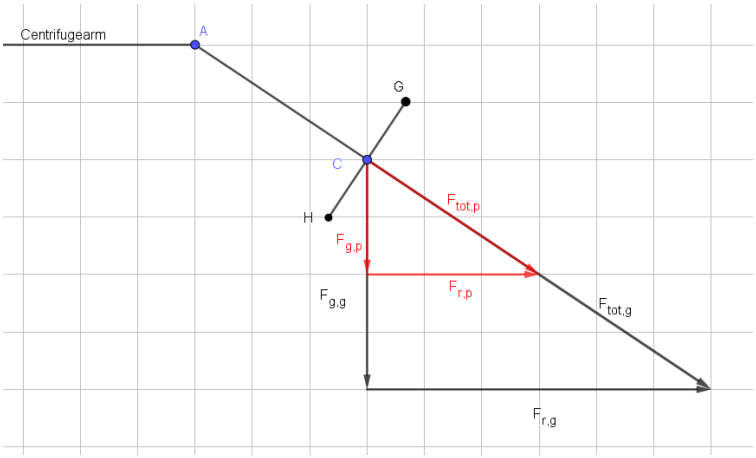


Figure 5: Schematic representation of the forces on a particle in the gondola. It shows the arm of the centrifuge, with the centre of mass of the gondola C connected to the hinge A. The particle is also in point C. Perpendicular to line AC is the surface of the lower electrode, denoted by line GH. It can clearly be seen that the total force on the gondola, $F_{tot,g}$, is parallel to AC, so that GH stays perpendicular to $F_{tot,g}$. Since the total force on the particle, $F_{tot,p}$ is parallel to $F_{tot,g}$, it is also still perpendicular to GH.

There are some important points that need to be marked in the setup. The points can be defined arbitrarily, but after making a choice, consistency is required as boundary conditions for several equations will be set on these points. The points are shown in figure 6.

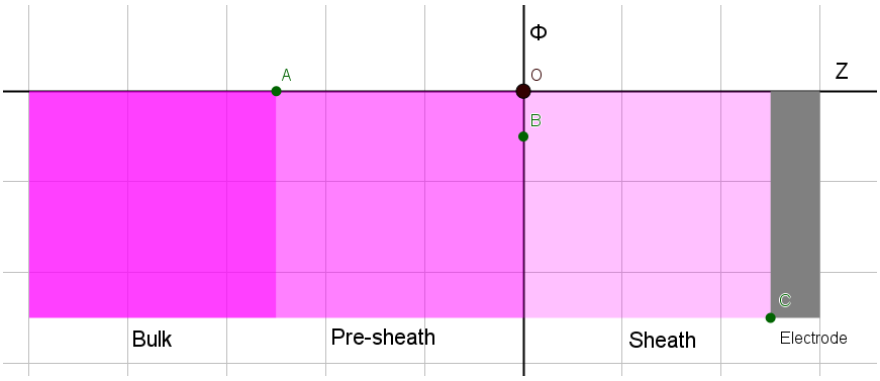


Figure 6: Image illustrating a few particular points with respect to position and potential in the plasma sheath. A: $z = -|s_0|$, $\Phi = 0$. B: $z = 0$, $\Phi = -|V_0|$. C: $z = s$, $\Phi = -|\Delta V|$.

With regards to the spatial dimension, a zero point and a positive direction have to be chosen. The zero point was chosen to be the sheath edge (more on that in subsection 2.3), and the electrode has a higher z-coordinate than the sheath edge. That means that a positive z points downwards in the real setup, and to the right in figure 6. Note that due to this choice, the force of gravity points in the positive z-direction. Furthermore, the potential had to be set to zero somewhere. For this, the plasma bulk was chosen. The plasma bulk was also assumed to have a constant potential, so that $\Phi = 0$ up until the pre-sheath edge.

2.3 Parameters

When trying to characterise the behaviour of a particle in a plasma sheath, the first question that comes to mind is: "Where is the sheath, actually?" In order to answer that, let's first look qualitatively at how a sheath

is formed. Regard a quasi-neutral hydrogen plasma where the ions and electrons have the same temperature. The thermal velocity of the ions and electrons is shown in equation 1 a and b respectively.

$$v_{T_i} = \sqrt{\frac{k_B T_i}{m_i}} \tag{1a}$$

$$v_{T_e} = \sqrt{\frac{k_B T_e}{m_e}} \tag{1b}$$

Since the hydrogen atoms are roughly 3600 times heavier than the electrons, their thermal velocity is about 60 times smaller. Since oftentimes the electron temperature is larger than the ion temperature, v_{T_e} is often even more than 60 times smaller than v_{T_i}. As a result, the electron flux into the plasma boundaries in a DC plasma (such as walls, electrodes and dust particles) is much larger than the ion flux. Hence the plasma boundaries acquire a negative charge. This negative charge in turn starts repelling electrons from the boundaries and attracting ions, gradually balancing out the electron and ion fluxes. An interesting side effect is that in the sheath there are far fewer electrons to recombine with the ions and release a photon, so the sheath appears dark.

The above explains a DC plasma sheath. In an RF plasma, sheaths form in a slightly different way. Because the electrons are so much lighter than ions, they react much more quickly to an applied electric field. The plasma frequency indicates the fundamental characteristic frequency of plasma oscillations. At the same time, it gives an indication of the highest-frequency oscillating electric field that the ions or electrons will respond to. The expressions and their values for the parameters in the setup are given for the ion and electron plasma frequency in equations 2a and 2b respectively.

$$\nu_{p,i} = \frac{1}{2\pi}\sqrt{\frac{e^2 n_0}{\epsilon_0 m_i}} = 0.592 MHz \tag{2a}$$

$$\nu_{p,e} = \frac{1}{2\pi}\sqrt{\frac{e^2 n_0}{\epsilon_0 m_i}} = 159.6 MHz \tag{2b}$$

Since the applied RF frequency ν_{rf} is 13.56 MHz, $\nu_{p,e} > \nu_{rf} > \nu_{p,i}$. This means that the electrons can easily follow the oscillations of the electrode, whereas the ions cannot. The ion bulk is thus roughly stationary, whereas the electron bulk oscillates rapidly. The electrons are continuously pushed to and pulled from the electrodes. Thus at the edges, the electron density averaged over several RF cycles is lower than $n_{e,0}$. This phenomenon is demonstrated in figure 7. As the electron cloud oscillates back and forth, the bulk density stays constant, whereas the sheath density drops whenever the electron cloud moves away. Hence the average electron density is lower at the edges than in the bulk. RF sheaths can be quantified analogously to the DC case. However, the model is designed for an RF plasma setup. Thus, in the derivation of the formulas to come, only the RF versions of formulas that differ for the DC and RF cases will be presented.

So in order to find the sheath thickness, we need to start by determining the electron and ion fluxes. But before that is possible, it is important to derive two important parameters that virtually every other parameter depends on to some degree; the electron and ion temperature.

2.3.1 Temperature profiles

Heat transfer in a plasma is significantly more complicated than in a single-molecule gas due to the presence of ions and electrons besides neutral atoms, which can all have different temperatures and velocities. For this reason, the temperature profiles will be based on geometric assumptions. The ion electrode and bulk temperatures and the electron bulk temperature are approximately known. These will be used as the boundary conditions for the temperature profiles. Furthermore, the ratio between the electron temperature and the ion temperature will be assumed constant. That ratio is equal to $T_{e,b}/T_b$ at the sheath edge, and is thus equal to that ratio everywhere in the sheath. Once T_i is found, the electron temperature can easily be derived from it. The boundary conditions in formula form are shown in equations 3.

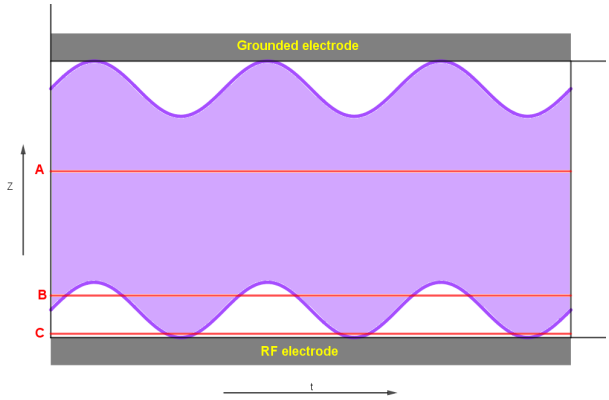


Figure 7: Electron density with an oscillating electron cloud. The horizontal axis shows the passage of time. The purple blob is the 1D electron cloud. At point A in the middle, the electron density seems constant. At B, the electron cloud goes away sometimes and the electron cloud almost never reaches C.

$$T_i(0) = T_n(0) = T_b \qquad (3a) \qquad T_i(s) = T_n(s) = T_{el} \qquad (3b) \qquad T_e(z) = T_i(z) \cdot T_{e,b}/T_b \qquad (3c)$$

For both ions and electrons, the three chosen possible profiles are constant, linearly decreasing from the electrode and exponentially decreasing from the electrode.

The constant profiles can be selected to test whether the temperature deviation is negligible. They are taken simply equal to the bulk temperature. This way they do violate the electrode boundary conditions, but violation of at least one boundary condition is inevitable with constant temperatures regardless of the choice. The bulk temperatures are known with more certainty, so it seems safer to use the bulk temperatures.

The possibility that the profile is linear is based on Fourier's law for heat conduction [10]:

$$\overline{q} = -k_T \nabla T \tag{4}$$

With $\overline{q}$ the heat flux and k_T the thermal conductivity. Since k_T is taken constant and $\overline{q}$ should be constant by energy conservation, ∇T should also be a constant. Hence, T itself would be linear. The linear ion temperature profile has the shape $T_{i,lin}(z) = A \cdot z + B$. From equation 3a follows that $B = T_b$. Using boundary condition 3b, it is then found that $A = (T_{el} - T_b)/sf$, so that $T_{i,lin}$ becomes

$$T_{i,lin} = (T_{el} - T_b)\frac{z}{sf} + T_b \tag{5}$$

$T_{e,lin}$ can be found by inserting 5 into 3c:

$$T_{e,lin} = (T_{el} - T_b)\frac{T_{e,b}}{T_b}\frac{z}{sf} + T_{e,b} \tag{6}$$

Since the thermal conductivity most likely depends on the temperature, either directly or through the gas density, an exponential temperature profile is expected to lie closer to reality. The exponential ion temperature profile has a shape $T_{i,exp}(z) = C \cdot e^{D \cdot z}$. Boundary condition 3a yields $C = T_b$. From equation 3b, $T_b \cdot e^{D \cdot s} = T_{el}$, which can be solved to find $D = ln(\frac{T_{el}}{T_b})\frac{1}{s}$. Simplifying the found expression gives:

$$T_{i,exp} = T_b \cdot \left(\frac{T_{el}}{T_b}\right)^{\frac{z}{s}} \tag{7}$$

Multiplying this solution by $\frac{T_{e,b}}{T_b}$ yields the profile for $T_{e,exp}$:

$$T_{e,exp} = T_{e,b} \cdot \left(\frac{T_{el}}{T_b}\right)^{\frac{z}{s}} \tag{8}$$

A big issue with these expressions is that they depend on the sheath thickness, whilst the sheath thickness - as will be shown later on - depends on various terms that depend on electron temperature. That means that neither could be calculated independently. For that reason, to determine the sheath thickness the average temperature will be used. It will be demonstrated shortly that the average temperatures are independent of the sheath thickness, so that the sheath thickness can be calculated, still taking into account the full temperature profile.

The definition of the average of a function $G(x)$ over the interval $[a, b]$ is given in equation 9.

$$G_{avg} = \frac{1}{b-a}\int_a^b G(x)dx \tag{9}$$

Naturally, in the constant temperature profiles the average temperature is equal to the constant temperature. For the linear profiles,

$$T_{i,a,lin} = \frac{1}{s}\left(\int_0^s ((T_{el}-T_b)\frac{z}{s}+T_b)dz\right) = \frac{1}{s}\left[\frac{T_{el}-T_b}{2s}z^2+T_b z\right]_0^s = \frac{T_{el}+T_b}{2} \tag{10}$$

So that with equation 3c $T_{e,a,lin}$ becomes

$$T_{e,a,lin} = \frac{T_{el}+T_b}{2}\frac{T_{e,b}}{T_b} \tag{11}$$

The exponential ion temperature profile average is shown in equation 12

$$T_{i,a,exp} = \frac{1}{s}\left(\int_0^s T_b\cdot\left(\frac{T_{el}}{T_b}\right)^{\frac{z}{s}}dz\right) = \frac{1}{s}\left[\frac{sT_b}{ln(\frac{T_{el}}{T_b})}\cdot(\frac{T_{el}}{T_b})^{\frac{z}{s}}\right]_0^s = \frac{T_{el}-T_b}{ln(\frac{T_{el}}{T_b})} \tag{12}$$

Then, finally, $T_{e,a,exp}$ can be found to be

$$T_{e,a,exp} = \frac{T_{el}-T_b}{ln(\frac{T_{el}}{T_b})}\frac{T_{e,b}}{T_b} \tag{13}$$

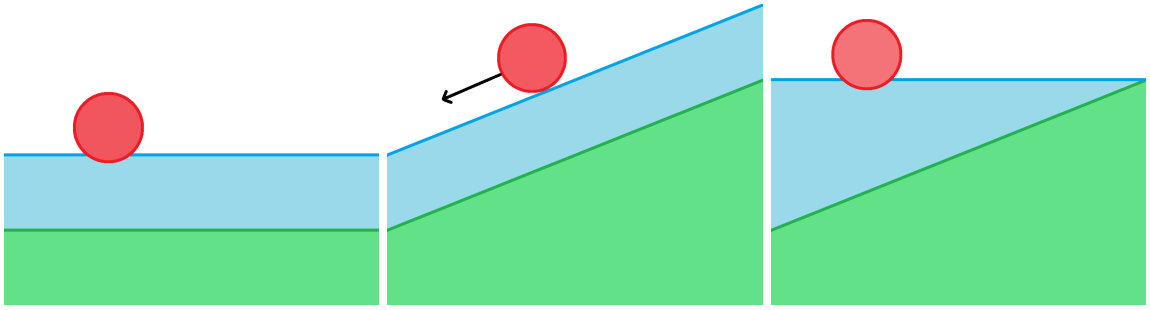
And indeed, none of the expressions 10 through 13 depends on s. Now that we're at it, let's also derive the expressions for the gradients of the various ion temperature profiles. In one dimension, the gradient is simply the derivative towards the spatial coordinate. Thus, for the constant ion temperature profile, ∇T_c=0, and for the linear profile $\nabla T_{lin} = \frac{T_{el}-T_b}{s}$. The exponential profile can be found as follows:

$$\nabla T_{exp} = \frac{\partial T_{exp}}{\partial z} = \frac{T_b ln\left(\frac{T_{el}}{T_b}\right)}{s}\left(\frac{T_{el}}{T_b}\right)^{\frac{z}{s}} \tag{14}$$

2.3.2 Length scales

There are several length scales present in the plasma that determine the way the particles move and how the electric fields behave. These will be derived here.

The first length scale is the Debye length. It appears as a consequence of the mass difference between electrons and ions. Being 3600 times lighter than ions, electrons will respond to electric fields much faster. This response ensures that no large-scale electric fields will exist in the plasma. Because if there was an electric field in the plasma, the electrons would move opposite the direction of the electric field and produce a net electric field in the opposite direction, to cancel the original field out. This can best be compared to a ball floating on water, as shown in figure 8. On an initially level surface covered in a layer of water, a floating ball will remain stationary. If one side of the surface is lifted to create a slope, the ball does not float down off the slope. Instead, the water reorganises to form a new, level surface, on which the ball remains stationary. The electron cloud will take the role of the water and the slope of the surface is the electric field.



(a) A ball on a level surface covered in water. (b) The surface just after creating a slope. (c) The surface after reshuffling of the water.

Figure 8: Schematic drawings of an object on a non-stationary surface.

This effect causes electric deviations to have a finite 'reach' in a plasma. The Debye length is thus the length scale over which electric fields lose their influence, and it follows quite naturally from the plasma equations. Maxwell's equations state that [11]

$$\nabla \cdot \overrightarrow{E} = \frac{\rho_c}{\epsilon_0} \tag{15}$$

Where ρ_c is the charge density. The electric field is equal to minus the gradient of the electric potential $\overrightarrow{E} = -\nabla\Phi$ and $\rho_c = e(n_i - n_e)$, the number of ions minus the number of electrons times their charge, where it was assumed that the ions lack only a single electron. Substituting this in equation 15 gives:

$$\nabla^2\Phi = \frac{e}{\epsilon_0}(n_e - n_i) \tag{16}$$

Lieberman et al. [12] derives an expression for the electron density as function of Φ and finds:

$$n_e = n_{e,0}e^{\frac{k_B\Phi}{eT_e}} \tag{17}$$

Then, assuming immmobile ions, the ion density is constant; $n_i = n_{i,0}$. Assuming quasi-neutrality in the plasma bulk, $n_{e,0}$ and $n_{i,0}$ can both be set to n_0. Filling all this in in the one-dimensional version of equation 16, the result is:

$$\frac{\partial^2\Phi}{\partial x^2} = \frac{en_0}{\epsilon_0}(e^{\frac{k_B\Phi}{eT_e}} - 1) \tag{18}$$

Performing a 1st-order series expansion of n_e around $\frac{k_B\Phi}{eT_e}$ yields $n_e = n_0(1 + \frac{k_B\Phi}{eT_e})$. Since performing a Taylor series expansion (while useful on small scales) is not effective on the order of the sheath, the Boltzmann expression is expected to appear in the best model. Substituting the Taylor expansion, equation 18 becomes

$$\frac{\partial^2\Phi}{\partial x^2} = \frac{en_0}{\epsilon_0}\frac{k_B\Phi}{eT_e} \tag{19}$$

Which is a second-order ordinary differential equation and can be solved easily to find:

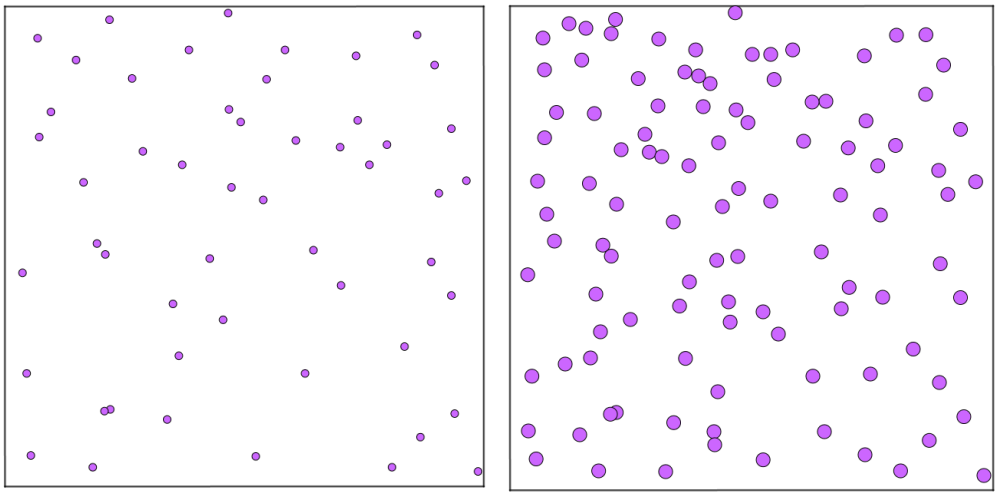
$$\Phi = \Phi_0 e^{-\frac{|x|}{\lambda_{D,e}}} \tag{20}$$

Where

$$\lambda_{D,e} \equiv \sqrt{\frac{\epsilon_0 k_B T_e}{e^2 n_e}} \tag{21}$$

Is the electron Debye length. The ion Debye length can be derived in a similar fashion. This time the electron density will be set constant and the ion density profile (and its taylor expansion) are given by:

$$n_i = n_0 e^{-\frac{k_B\Phi}{eT_i}} \approx n_0\left(1 - \frac{k_B\Phi}{eT_i}\right) \tag{22}$$



(a) Dilute gas. (b) Denser gas.

Figure 9: Frontview of two gases of different density and atom size.

Filling this in in equation 16 and solving for Φ, the ion Debye length rolls out:

$$\lambda_{D,i} \equiv \sqrt{\frac{\epsilon_0 k_B T_i}{e^2 n_i}} \tag{23}$$

These two Debye lengths can be combined to find a 'total Debye length' by using the relation [13]

$$\frac{1}{\lambda_D^2} = \frac{1}{\lambda_{D,i}^2} + \frac{1}{\lambda_{D,e}^2} \tag{24}$$

so that

$$\lambda_D = \frac{\lambda_{D,i}\lambda_{D,e}}{\sqrt{\lambda_{D,i}^2 + \lambda_{D,e}^2}} \tag{25}$$

Another important length scale determines the distance that atoms, ions or electrons can travel before bumping into something. This length scale is the mean free path. Imagine being an electron incident on a cloud of gas. Your view of the gas cloud will be as shown in figure 9, depending on the density of the gas and size of the gas atoms. It is clearly visible that in the more dilute gas with small atoms 9a, there is much more room to pass through the particles unscathed than in the dense gas 9b. This goes to show that the distance a particle can travel before hitting something decreases as both the particle frontal area and the gas density increase. The mean free path determines what the type of the considered sheath is. If $\lambda_{mf,i} >> s$, most ions will pass through the sheath without colliding with any atom at all and can follow a straight path. If $\lambda_{mf,i} << s$, almost all ions will collide with atoms and bounce around like billiard balls rather than follow a straight path to the electrode. In the latter case, the ions (as well as electrons) are much less mobile and will thus shield the bulk over a smaller region. As a result, the collisional sheath is thinner than the collisionless sheath. This qualitative explanation will be deemed sufficient to pose without further derivation that the mean free path is equal to:

$$\lambda_{mf} = \frac{1}{n_g \sigma} \tag{26}$$

The formal derivation of equation 26 can be found in the book of Lieberman et al. [12]. The cross section σ is in this case the momentum transfer cross section. It can in a simplified case be taken as the literal cross-section of the gas atoms, called the 'hard sphere approximation'. In this case, a collision will occur if the centres of the particles come closer together than the sum of their radii - say that distance is a - and

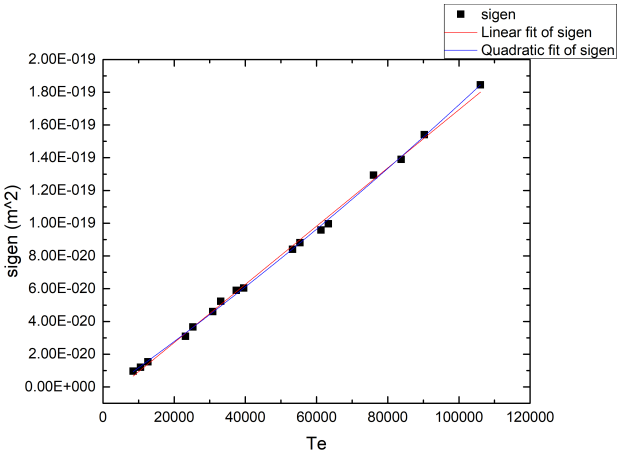


Figure 10: Linear (red) and second order polynomial (blue) fits to the electron-neutral scattering cross section as function of electron temperature.

then σ will be equal to πa^2. In practice, σ depends on many parameters such as the temperature and the type of particle used. Phelps [14] derived an expression for the ion-neutral momentum transfer cross section in argon, which is shown in equation 27.

$$\sigma_{i,n} = \frac{2 \cdot 10^{-19}}{(\frac{k_B T_i}{e})^{0.5}(1 + \frac{k_B T_i}{e})} + \frac{3 \cdot 10^{-19} \frac{k_B T_i}{e}}{(1 + \frac{k_B T_i}{3e})^{2.3}} \tag{27}$$

Taylor expanding this expression to second order greatly simplifies it:

$$\sigma_{i,n} = 2.32022 \cdot 10^{-18} - 6.076 \cdot 10^{-17} \frac{k_B T_i}{e} + 7.04191 \cdot 10^{-16} \frac{k_B T_i}{e}; \tag{28}$$

Furthermore, Paeva [15] uses the constant value $8 \cdot 10^{-19} m^2$ for the ion-neutral scattering cross section. For the electron-neutral scattering cross section, Subramanian and Kumar [16] collected data for electron temperatures between 10,000 and 100,000 K. $\sigma_{e,n}$ is set out against T_e in figure 10, along with a linear and quadratic fit against the data. The linear fit was chosen because at first glance the connection seemed to be linear. However, after making the linear fit, a slight upward curvature was seen, which implied that a quadratic term would improve the result. This turned out to be the case, as the residual sum of squares of the linear fit was double that of the quadratic fit. The linear fit resulted in equation 29

$$\sigma_{e,n} = -8.49694 \cdot 10^{-21} + 1.77793 \cdot 10^{-24} \cdot T_e \tag{29}$$

And the quadratic fit

$$\sigma_{e,n} = -3.80964 \cdot 10^{-21} + 1.53036 \cdot 10^{-24} \cdot T_e + 2.30868 \cdot 10^{-30} \cdot T_e^2 \tag{30}$$

Considering the relatively small deviations in both electron and ion temperature as compared to the order of magnitude of the numbers in equations 27 through 30, not much of a difference between the two expressions for $\sigma_{i,n}$ and $\sigma_{e,n}$ is expected. The constant ion-neutral scattering cross section is expected not to be very representative of reality, as Paeva [15] used it second-hand for the plasma parameters relevant in that particular case, and perhaps not for the parameters of this thesis.

2.3.3 Potential

The sheath is the part of the plasma where the quasi-neutrality is broken to shield the plasma bulk from external electric fields. It is tempting to then define the sheath edge at the point where the potential becomes

non-zero. However, one soon finds that in practice the potential is not entirely constant in the bulk, so that the only point where the potential is exactly zero is right in the middle where it was defined to be zero. The question is then: Where does the sheath begin? This problem was solved by Bohm in 1949 [17]. The most simple derivation is as follows. It is assumed that $\Phi(0) = 0$ and that quasi-neutrality holds up until $z = 0$. Furthermore the assumption is made that the plasma is collisionless and there is no net ionization, so that ion energy and flux are conserved. Ion energy consists of the sum of kinetic and potential energy. Equation 31 equates the ion energy at $z = 0$ and in the sheath.

$$\frac{1}{2}m_i u_i^2(0) + 0 = \frac{1}{2}m_i u_i^2(z) + e\Phi(z) \tag{31}$$

And flux conservation dictates that

$$n_{i,0}u_i(0) = n_i(z)u_i(z) \tag{32}$$

Substituting 32 into 31 and solving for $n_i(z)$ yields

$$n_i(z) = n_{i,0}\left(1 - \frac{2e\Phi(z)}{m_i u_i^2(0)}\right)^{-\frac{1}{2}} \tag{33}$$

Using again Boltzmann distributed electrons as in equation 17, setting $n_{i,0} = n_{e,0} = n_0$ and substituting the expressions for n_i and n_e into equation 16, the differential equation 34 follows:

$$\frac{d^2\Phi}{dz^2} = \frac{en_0}{\epsilon_0}\left(e^{\frac{k_B\Phi}{eT_e}} - \left(1 - \frac{2e\Phi(z)}{m_i u_i^2(0)}\right)^{-\frac{1}{2}}\right) \tag{34}$$

It can be proven that this equation only has real solutions if

$$u_i(0) \equiv u_{i,B} >= \sqrt{\frac{k_B T_e}{m_i}} \tag{35}$$

Where the ion velocity at the sheath edge was defined to be $u_{i,B}$. Equation 35 is called the Bohm criterion. Riemann [18] performs a rigorous mathematical analysis of it and when it is applicable. He explains that the physical reason the criterion exists is that the ions need sufficient velocity to be able to separate themselves from the electrons. If this is not possible, the sheath cannot form, which mathematically manifests itself in the form of a non-real potential.

In order to fulfill the Bohm criterion, the ions need to be accelerated beforehand. This is done in what is known as the 'pre-sheath'. Here the potential drops a little, although quasi-neutrality can be assumed to hold. Using conservation of energy, the potential drop can be determined:

$$e\Phi(0) = \frac{1}{2}m_i u_{i,B}^2 \tag{36}$$

Where the ion bulk energy was assumed to be 0. Substituting the bohm criterion for $u_i^2(0)$ in equation 36, setting $\Phi(0) = V_0$ and solving for V_0 gives:

$$V_0 = \frac{k_B T_e}{2e} \tag{37}$$

Using that in steady state the ion flux as well as the electron flux are constant throughout the sheath and equal to each other, the ion flux at the sheath edge (38) can be equated to the electron flux at the wall (39).

$$\Gamma_i = n_i(0)\sqrt{\frac{k_B T_e}{m_i}} \tag{38}$$

$$\Gamma_e = n_e(0)e^{\frac{e\Phi(s)}{k_B T_e}}\sqrt{\frac{8k_B T_e}{\pi m_e}} \tag{39}$$

Setting the fluxes equal and rewriting slightly gives

$$\frac{n_i(0)}{n_e(0)}\sqrt{\frac{k_B T_e}{m_i}} = \sqrt{\frac{8k_B T_e}{\pi m_e}}e^{\frac{e\Phi(s)}{k_B T_e}} \tag{40}$$

Assuming quasi-neutrality in the pre-sheath so that $\frac{n_i(0)}{n_e(0)} = 1$, equation 40 can be solved for $\Phi(s)$ to find equation 41.

$$\Phi(s) = -\frac{k_B T_e}{2e} ln\left(\frac{m_i}{2\pi m_e}\right) \tag{41}$$

It should be noted that because of the quasi-neutrality assumption used to derive $\Phi(s)$, $\Phi(s)$ is slightly offset. By making that assumption, $\Phi(0)$ was implicitly assumed to be zero, which the Bohm criterion does not allow. To find the potential difference between the bulk and wall ΔV, V_0 is added to the found expression for $\Phi(s)$ so that

$$\Delta V = -\frac{k_B T_e}{2e}\left(ln\left(\frac{m_i}{2\pi m_e}\right) + 1\right) \tag{42}$$

Which for argon takes the value of approximately $-5.2\frac{k_B T_e}{e}$.

To find the potential profile, Lieberman et al. [12] makes a full derivation of the Child law sheath. This is a thin, planar sheath - like the one at the electrode in this setup - in which the ion energy is assumed small compared to Φ and the ion flux is assumed constant. The potential found is

$$\Phi(z) = \Delta V \cdot \left(\frac{z}{s}\right)^{\frac{4}{3}} \tag{43}$$

However, from equation 43 follows that $\Phi(0) = 0 < |V_0|$ and so the Bohm criterion is violated. This is a consequence of the fact that Lieberman et al. [12] uses slightly different boundary conditions. Therefore, equation 43 will be rewritten as a function $\Phi(z) = A + B \cdot z^{4/3}$, with the boundary conditions $\Phi(0) = V_0$ and $\Phi(s) = \Delta V$. $\Phi(0) = A + B \cdot 0 = A = V_0$, and so $\Phi(s) = V_0 + B \cdot s^{4/3} = \Delta V$. This can be solved to find $B = \frac{\Delta V - V_0}{s^{4/3}}$. The Child law potential complying with the boundary conditions in this setup thus becomes

$$\Phi(z) = V_0 + (\Delta V - V_0)\left(\frac{z}{s}\right)^{\frac{4}{3}} \tag{44}$$

Furthermore, numerical calculations have been performed to find the potential profile in the sheath. Two functions have been fitted to this profile, a polynomial of 3rd order, which was found to be

$$\Phi(z) = 4922.61401 - 436128.7157 \cdot z^* + 12879600 \cdot (z^*)^2 - 126864000 \cdot (z^*)^3 \tag{45}$$

Where $z^* \equiv (z - (snum - 0.04))$ results from the fact that the numerical calculations were performed over the entirety of the plasma and used the grounded electrode for $z = 0$. It amounts to a translation by 4 cm minus the sheath thickness. And a root function of the form $C \cdot (A - z^*)^B + D$:

$$\Phi(z) = 1171.94967 \cdot (0.0381 - z^*)^{0.78531} - 13.4034 \tag{46}$$

At an electron temperature of 30000 K, the wall potential for the child law potential is about 13 Volts. For an 80 V RF potential, 13 Volts total potential difference seems very little. The numerical profile has a ΔV of 50 Volts, which seems much more reasonable. Since the residual sum of squares for the root function fitting is over 4000 times larger than for the polynomial, the polynomial fitted function for Φ is likely the best fit for the model.

The electric fields can be derived trivially from the potentials, as $E = -\nabla\Phi = -\frac{d\Phi}{dz}$ in one dimension. The Child law electric field is thus

$$E = -\frac{4}{3} \cdot \frac{\Delta V - V_0}{s^{4/3}} \cdot z^{\frac{1}{3}} \tag{47}$$

and the electric fields derived from the numerical potentials fitted to a polynomial and a root are given in equations 48 and 49, respectively.

$$E = 436129 - 25759200 \cdot z^* + 380592000 \cdot (z^*)^2 \tag{48}$$

$$E = 920.344 \cdot (0.03801 - z^*)^{-0.21469} \tag{49}$$

Lastly, another expression for the ion density can be derived based on the Child law. In one dimension,

$$\frac{dE}{dz} = \frac{\rho_c}{\epsilon_0} = \frac{e}{\epsilon_0}(n_i - n_e) \approx \frac{en_i}{\epsilon_0} \tag{50}$$

Where the last approximate equality holds under the assumption that $n_e << n_i$. This is reasonable in the sheath. The ion density based on Child's law is then given by equation 51

$$n_i = \frac{\epsilon_0}{e}\frac{dE}{dz} = -\frac{4\epsilon_0}{9el}\frac{\Delta V - V0}{s^{4/3}} \cdot z^{-\frac{2}{3}} \tag{51}$$

The Child law ion density is based on an approximation and built on a series of assumptions necessary for Child's law. It is thereby expected to be less appropriate than equation 33, as that ion density profile was based on the two much more basal principles of energy and flux conservation.

The last relevant potential is that of the particle, as V_p influences the particle charge. See the thesis of Paeva [15] for the full derivation. Here only the general idea and the outcome are presented. Like the electrodes, the dust particle also forms a sheath around it. This sheath is on the order of the Debye length of the plasma, and if the particle radius is much smaller than the Debye length, the sheath will be relatively thick and the so-called Orbit Motion Limited (OML) theory can be applied. It assumes a collisionless particle sheath and that ions are collected if their trajectories reach the surface of the dust particle. In steady state, the particle charge does not change anymore, so the net current to the particle is zero. That means that the ion and electron currents are equal. Calculating and equating them gives the expression

$$e^{\frac{eV_p}{k_B T_e}} = \sqrt{\frac{T_i m_e}{T_e m_i}}\frac{n_i}{n_e}\left(1 - \frac{eV_p}{k_B T_i}\right) \tag{52}$$

Solving for V_p gives

$$V_p = \frac{k_B}{e}\left(- T_e W\left[\sqrt{\frac{T_i m_i}{T_e m_e}}\frac{n_e}{n_i}e^{\frac{T_i}{T_e}}\right] + T_i\right) \tag{53}$$

Where $W[x]$ denotes the lambert W function, also known as the 'ProductLog', which does not have an actual expression. It is simply defined as the solution for x in the equation $z = x \cdot e^x$ [19]. It does have a numerical expression that MatLab can work with, so no further simplification of equation 53 is necessary.

The total charge of the particle Q_p is equal to the surface area of the spherical particle times the surface charge density,

$$Q_p = 4\pi r_p^2 \sigma_c \tag{54}$$

The surface charge density of a particle can be derived from Maxwell's equations [11] and is equal to

$$\sigma_c = \epsilon_0 E_r(r_p) = -\epsilon_0 \frac{dV_p}{d_r} \tag{55}$$

Outside a spherical object, $V \propto \frac{1}{r}$, so $\frac{dV_p}{dr} = \frac{-V_p}{r}$. Combining this result with equations 54 and 55 gives

$$Q_p = 4\pi r_p \epsilon_0 V_p \tag{56}$$

2.3.4 Sheath thickness

Now that the necessary ingredients - the sheath edge potential and the Debye length - have been determined, it can be posed without further derivation that the Child law sheath thickness is

$$s = \frac{10}{9\sqrt{3}}\lambda_{D,e}\left(\frac{\Delta V}{T_e}\right)^{\frac{3}{4}} \tag{57}$$

for a collisionless RF plasma. This was derived by Lieberman et al. [12], who also demonstrates that for a DC plasma, the sheath becomes $\sqrt{\frac{27}{50}}$ times thinner. Furthermore, Lieberman et al. [12] derives an expression

for the thickness of the collisional sheath with the assumption that the ion mean free path is constant, which is given in equation 58,

$$s = 1.155 \frac{-(\frac{e\Delta V}{k_B T_e})^{\frac{3}{5}}}{(\frac{v_{i,B}}{v_{T_i}})^{\frac{2}{5}}(\frac{\lambda_D}{\lambda_{mf,i}})^{\frac{1}{5}}} = 9388.2 \cdot \left(\frac{\Delta V^3}{n_e^2 n_g \sigma_{i,n} T_e}\right)^{\frac{1}{5}} \tag{58}$$

as well as an expression for the collisional sheath with constant ion mobility, shown in equation 59.

$$s = 129.957 \cdot \left(\frac{\Delta V^4 n_e}{T_e^5 n_g^2 \sigma^2}\right)^{\frac{1}{6}} \tag{59}$$

For the numerical potential profiles, the sheath thickness was also determined numerically. The sheath starts at the point where $\Phi = V_0$, which can simply be deduced from the numerical potential. The exact method will be explained in section 3.

2.3.5 Velocities

The velocities of ions can be derived from conservation of energy. The kinetic energy gained by an ion is equal to the electrostatic potential energy lost. Assuming the ions leave the plasma bulk with zero velocity, their energy balance is given by equation 60.

$$\frac{1}{2} m_i u_i^2(z) = -e\Phi(z) \tag{60}$$

Solving equation 44 for u_i gives

$$u_i(z) = \sqrt{-\frac{2e}{m_i}\Phi} \tag{61}$$

To determine if the velocity increase of ions is negligible, the Bohm velocity can also be selected as a velocity profile. Naturally, the ions are expected to be accelerated towards the electrode so their velocity is not constant, and hence the velocity profile from 61 is expected rather than the Bohm velocity.

The thermal velocities of ions and electrons have been given in equation 1, and the neutral thermal velocity can be expressed completely analogously as:

$$v_{T_n} = \sqrt{\frac{k_B T_n}{m_n}} \tag{62}$$

This, however, is the expression for the thermal velocity in 1 dimension assuming that particles can only move in 1 dimension. Although this model is 1-dimensional, the setup is not, and so the particle thermal velocity is not necessarily in the 1 dimension here regarded. To take into account only the z-components of 3-dimensional velocities, the root mean squared thermal velocity is used:

$$v_{T_n} = \sqrt{\frac{2k_B T_n}{\pi m_n}} \tag{63}$$

The ion and electron thermal velocities can yet again be calculated analogously.

An effective ion velocity can be calculated which accounts for both the ion drift and thermal velocities:

$$u_{i,ef} = \sqrt{u_i^2 + v_{T_i}^2} \tag{64}$$

The neutral drift velocity determines the magnitude of the neutral drag force. It is expected that the neutral drag force is negligible. In order to prove that later on, a maximum neutral drift velocity will be determined now. If the neutral drag force is negligible even with this maximum velocity, it can be neglected altogether. The gas inlet is set up to send a volume flux Γ_n of 1 sccm (cm^3/min for standard temperature and pressure,

300 K and 1013 hPa) of argon gas into the plasma chamber. This amounts to $1/60 \cdot 10^{-6}\ m^3/s$. The gas outlet is circular in shape and has a diameter of $6.3 \cdot 10^{-2}$ m. To determine the maximum neutral velocity, it is assumed that all this gas flows straight down upon the dust particle and then on to the outlet, as if it were flowing in a laminar fashion through a tube with a diameter of 6.3 cm. Since the gas volume flux is equal to the gas velocity times the area through which the gas flows, the velocity can be determined with the formula

$$u_n = \frac{\Gamma_n}{\pi(diameter/2)^2} = \frac{10^{-6}}{\pi(0.063/2)^2 \cdot 60} = 5.344 \cdot 10^{-6} \tag{65}$$

Because the forces on the dust particle will be determined at equilibrium positions, the particle velocity will be assumed to be equal to zero: $u_p = 0$.

2.4 Forces

In this subsection, the forces acting on the dust particle will be quantified. Five forces will be considered: Gravity, neutral drag, ion drag, coulomb and the thermophoretic force. These five forces are used because they are assumed to be most significant. Other forces acting on the particle that will not be considered are the Lorentz force, because there is no magnetic field in the plasma, the electron drag, because electrons are much lighter than ions, and the rocket and radiation pressure forces because they are much smaller than the five forces initially mentioned [20].

2.4.1 Gravity

The first force to be considered is the force of gravity. As mentioned in subsection 2.2, the perceived force of gravity on an object with mass m is

$$\overrightarrow{F_{g^*}} = m \cdot \overrightarrow{g^*} \tag{66}$$

The mass of the used particle is equal to

$$m_p = density \cdot volume = \rho_p \cdot \frac{4}{3}\pi r_p^3 \tag{67}$$

So that the perceived force of gravity becomes

$$\overrightarrow{F_{g^*}} = \frac{4}{3}\pi r_p^3 \cdot \rho_p \cdot \overrightarrow{g^*} \tag{68}$$

From experiments conducted with the centrifuge, equilibrium heights for the particle at various values of g^* were determined to obtain figure 11. Two functions were fitted against this data; a 3rd-order polynomial and a function of the form $y = C \cdot (A - x)^B$, shown in figure 11. These fitted functions $z(g^*)$ were then inverted to obtain two functions $g^*(z)$, which can be plotted like all other variables. Because F_{g^*} was plotted as function of the equilibrium height of the particle, the particle was stationary at every point on this function, i.e. the net force on the particle was zero. This means that at every point on this function, the sum of the 4 remaining forces should be equal in magnitude but opposite in direction to F_{g^*}. Using this, the model can be tested for quality.

2.4.2 Neutral drag force

The neutral drag force originates from collisions with neutral atoms. There are two manifestations of this force. The first occurs if the bulk velocity of the atoms is large with respect to the particle's velocity (i.e. the fluid moves around the particle). As atoms strike the particle, they transfer some of their momentum to the particle. Because it is not one big particle imparting a lot of momentum at once, but many small atoms imparting a little bit of momentum continuously, the particle is gradually accelerated in the direction of movement of the atoms. If the bulk velocity of the atoms is small with respect to the particle's velocity (the

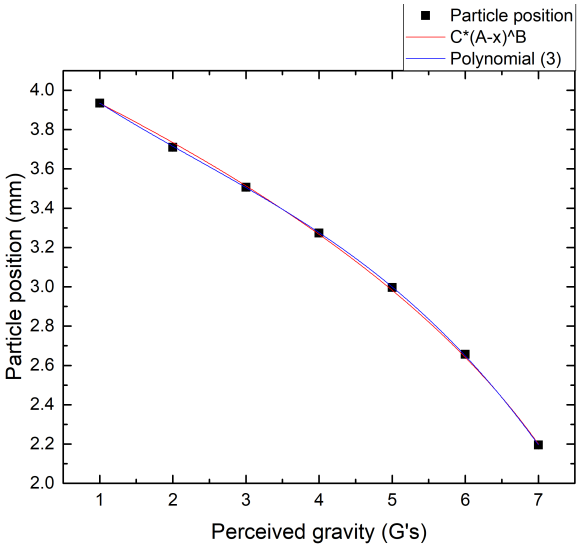


Figure 11: Equilibrium height of the particle as function of perceived gravity, fitted against the function $y = C \cdot (A - x)^B$ (red) and a 3rd-order polynomial (blue).

particle moves through the fluid), the particle has to push atoms aside in order to move through the fluid. For this, it has to give the atoms in the fluid momentum, thus decreasing its own, so that it experiences a retarding force. These two manifestations indicate that relative velocity is important. Furthermore it is important to consider whether the relative velocity of the particle is large with respect to the thermal velocity of the neutral particles, so whether:

$$|\overrightarrow{u_p} - \overrightarrow{u_n}|/v_{T_n} >> 1 \tag{69}$$

By approximating the particle and the atoms as hard spheres and assuming that $u_p = 0$ so that $|u_p - u_n| = |u_n|$, Lieberman et al. [12] finds:

$$\overrightarrow{F_{n,Lieb}} = \pi r_p^2 m_n v_{T_n} n_g \overrightarrow{u_n} \tag{70}$$

This can be regarded as the product of the particle area, atom momentum and atom flux. Using an expression of the drag force for a Maxwellian distribution of the gas molecules, Bouchoule [13] shows that for a low relative velocity and pure diffuse reflection (meaning that atoms bounce off the particle at all angles):

$$\overrightarrow{F_{n,bouch,s}} = -\frac{4}{3}\pi r_p^2 m_n n_g v_{T_n} (\overrightarrow{u_p} - \overrightarrow{u_n}) \tag{71}$$

And for a high relative velocity:

$$\overrightarrow{F_{n,bouch,f}} = -\pi r_p^2 m_n n_g |\overrightarrow{u_p} - \overrightarrow{u_n}| (\overrightarrow{u_p} - \overrightarrow{u_n}) \tag{72}$$

Although the vanderWaals-forces through which the argon atoms interact with each other are very weak, they do not come into play only on contact of two atoms. Hence the hard sphere approximation is just that: an approximation. Furthermore, since the gas in the setup is on the order of magnitude of room temperature, the fast particle approximation probably fails, and 71 will be expected.

2.4.3 Ion drag force

The ion drag force is a force very similar to the neutral drag force. Drifting ions will also impart momentum onto the particle. However, because the positively charged ions interact electrically with the negatively

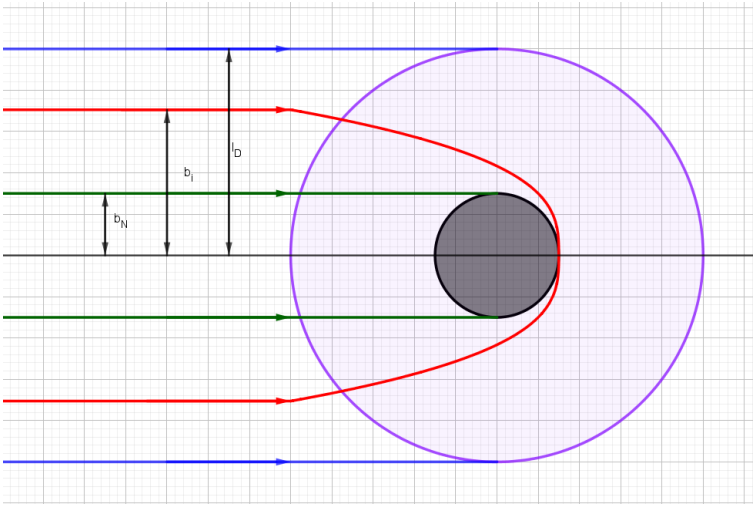


Figure 12: Schematic representation of the impact parameters of ions and neutral atoms incident on a particle. The particle (black) has an electrostatic reach (purple). Neutral atoms will be collected if they come in from between the green lines; $b < r_p$. Ions will be collected if they approach the particle from in between the red lines, $b < b_{coll}$, and will still transfer some momentum if they come in from between the blue lines; $b < \lambda_D$.

charged particle, more ions will be collected by the particle, and ions that pass by the particle in close proximity also transfer some momentum to the particle ('ion orbital drag'). To quantify how close two particles approach each other, the impact parameter b is used. If a particle approaches another particle along a straight line, its impact parameter is the shortest distance between that line and the centre of the other particle. The largest impact parameters for neutral collection, ion collection and ion orbital drag are shown in figure 12. To find the total ion drag, both interactions have to be considered. The collection drag is rather simple. By drawing a circle with radius b_{coll} around the particle, the collection area, called the 'collection cross section' (σ_{coll}) of the particle can be found, as Paeva [15] did:

$$\sigma_{coll} = \pi \cdot b_{coll}^2 = \pi r_p^2 \cdot (1 - \frac{2eV(r_p)}{m_i u_i^2}) \tag{73}$$

Assuming that colliding atoms transfer all their momentum to the particle, the ion collection drag force can be found analogously to the neutral drag force:

$$\overrightarrow{F_i^{coll}} = \sigma_{coll} \cdot p_i \cdot \Gamma_i = \pi r_p^2 \cdot (1 - \frac{2eV(r_p)}{m_i u_i^2}) \cdot m_i \overrightarrow{u_i} \cdot n_i u_i \tag{74}$$

The orbital drag requires more care, since the ions contributing to this force transfer varying fractions of their momentum to the particle, depending on their impact parameter. Paeva [15] also demonstrates the integral to be solved in the case of mono-energetic ions and a coulomb potential that is cut off at a distance $r > \lambda_D$ from the particle:

$$\sigma_{orb} = 4\pi \int_{b_{coll}}^{\lambda_D} \frac{2b\mathrm{d}b}{1 + (\frac{b}{b_0})^2} \tag{75}$$

Where $b_0 = r_p \frac{-eV(r_p)}{m_i u_i^2}$. This is the impact parameter for which the ions scatter at an angle $\pi/2$. Solving the integral in equation 75 gives equation 76:

$$\sigma_{orb} = 4\pi b_0^2 ln\left(\frac{\lambda_D^2 + b_0^2}{b_{coll}^2 + b_0^2}\right) \tag{76}$$

The logarithm in equation 76 is called the 'Coulomb logarithm'.

The orbital ion drag force can also be found analogously to the neutral drag force, with σ_{orb} as the cross section.

$$\overrightarrow{F_{i,paeva}^{coll}} = \sigma_{orb} \cdot p_i \cdot \Gamma_i = 4\pi b_0^2 ln\left(\frac{\lambda_D^2 + b_0^2}{b_{coll}^2 + b_0^2}\right) \cdot m_i \overrightarrow{u_i} \cdot n_i u_i \tag{77}$$

Kilgore et al. [21] derives a different expression for the coulomb logarithm, resulting in an orbital ion drag force:

$$\overrightarrow{F_{i,kill}^{coll}} = 2\pi b_0^2 ln\left(1 + \frac{\lambda_D^2}{b_0^2}\right) \cdot m_i \overrightarrow{u_i} \cdot n_i u_i \tag{78}$$

Both these expressions assume mono-energetic ions. For poly-energetic ions, Khrapak et al. [22] has derived another different coulomb logarithm, which leads to the orbital ion drag force as shown in equation 79. Accounting for a larger range of ions, this expression for the ion drag force will likely be the most accurate.

$$\overrightarrow{F_{i,khrap}^{coll}} = 4\pi b_0^2 ln\left(\frac{\lambda_D + b_0}{r_p + b_0}\right) \cdot m_i \overrightarrow{u_i} \cdot n_i u_i \tag{79}$$

2.4.4 Coulomb force

Since the particle acquires a charge and there is an electric field in the sheath, the particle experiences an electrostatic force given in equation 80 [11].

$$\overrightarrow{F_{C,mon}} = Q_p \cdot \overrightarrow{E} \tag{80}$$

This equation assumes that the charge is distributed evenly across the surface of the particle, so that the charge behaves as a monopole. However, the mere presence of the particle causes polarization in the plasma. Fortov et al. [23] accounts for this polarization by replacing the electric field by an effective electric field, which includes a correction term shown in 80.

$$\overrightarrow{E_{eff}} = \overrightarrow{E}\left(1 + \frac{(r_p/\lambda_D)^2}{3(1 + r_p/\lambda_D)}\right) \tag{81}$$

Furthermore, this plasma polarization induces a dipole moment $\overrightarrow{p} = r_p^3 \cdot \overrightarrow{E_{eff}}$ in the particle, which adds an additional term to the total coulomb force with magnitude:

$$\overrightarrow{F_{C,dip}} = (\overrightarrow{p}\nabla)\overrightarrow{E} = r_p^3 \cdot \overrightarrow{E_{eff}}\nabla \cdot \overrightarrow{E} \tag{82}$$

Referring back to equation 15, bearing in mind that $\rho_c = e(n_i - n_e)$, $\nabla \cdot \overrightarrow{E} = e(n_i - n_e)$, we can substitute for $\overrightarrow{E_{eff}}$ and $\nabla \cdot \overrightarrow{E}$ in equation 82 to get equation 83 for the dipole term of the coulomb force:

$$\overrightarrow{F_{C,dip}} = r_p^3 e(n_i - n_e) \cdot \overrightarrow{E}\left(1 + \frac{(r_p/\lambda_D)^2}{3(1 + r_p/\lambda_D)}\right) \tag{83}$$

The total coulomb force is the sum of the monopole and dipole terms. Since adding the dipole term accounts for more physical phenomena than using just the monopole term does, the expectation is that including the dipole yields a more accurate expression for F_C.

2.4.5 Thermophoretic force

The final force considered is the thermophoretic force. Heat can be regarded as kinetic energy on a molecular level (the hotter a substance, the faster the molecules in the substance move). So, the molecules in a hotter (part of a) gas also carry more momentum than in a colder one. Imagine a gas of uniform density, with a very steep temperature drop somewhere in the middle. Introducing a balloon into the gas at that temperature drop, the gas on one side of the balloon will be hotter than on the other side. This means that the molecules

striking the balloon on one side will carry more momentum than on the other side, which results in a net force away from those molecules, towards the colder side of the gas. Talbot et al. [24] derives the following expression for the thermophoretic force:

$$\overrightarrow{F_{th}} = -\frac{32}{15}\frac{r_p^2}{v_{T_n}}\left(1 + \frac{5\pi}{32}(1-\alpha)\right)k_T \nabla T_n \tag{84}$$

Herein k_T is the thermal conductivity coefficient, taken to be equal to the k_T of argon, 0.017 W/mK [25]. ∇T_n is the neutral temperature gradient and α is the thermal accommodation coefficient, which determines the fraction of colliding molecules that are diffusely reflected. For $T_n < 500$K, α can be taken to be equal to 1, so that the thermophoretic force becomes:

$$\overrightarrow{F_{th}} = -\frac{32}{15}\frac{r_p^2}{v_{T_n}}k_T \nabla T_n \tag{85}$$

Aussems et al. [20] derives an expression which is very similar save for a constant:

$$\overrightarrow{F_{th}} = -\frac{8\sqrt{2\pi}}{15}\frac{r_p^2}{v_{T_n}}k_T \nabla T_n \tag{86}$$

It is not specified under what conditions this expression applies. However, Fortov et al. [23] derives an expression for the thermophoretic force for very fast ions, which - if rewritten - is exactly the same as equation 86. Hence equation 85 is expected to be the best option for the thermophoretic force.

3 Model description

In this section, a brief outline of how the model works will be given. It is a series of matlab scripts, of which the full code can be found in appendix A and the scripts can also be retrieved from `\\physstor\epg-common\Experimental_Data\Centrifuge\HyperGravityExperiments\ForceModel_BEPWouter\Matlab`.
There is a manual available in this folder as well, called fw#manual, where the # indicates the version of the manual. It is slightly more elaborate than what is denoted here, as it explains how to expand the scripts on top of just how they work. The different subsections denote the different scripts in the order they can best be executed in. When 'the current version' is mentioned, that refers to the latest version at the time this thesis was finished. Subsection 3.1 explains a script that allows incorporation of numerical data into the model. The script in subsection 3.2 furthers that process. selectvars gives prompts to select different options for different variables, and forcesworking incorporates these options. These four scripts can also be run in a row by running initialrun. The scripts in subsections 3.5 and 3.6 give two different ways to compare different versions of the model, and the one in subsection 3.7 helps analyse the data of the latter.

3.1 Loadnumerics

Loadnumerics extracts the data from simulations so that it can be incorporated in the model. In the first few lines it loads the data one by one (not all simultaneously), because all data is stored as 'TempData'. Thus, for each parameter's data, the script first saves this TempData as the appropriate variable before moving on to the next parameter. Furthermore, in the simulations, the wall potential was set to zero, rather than the bulk potential. To set the bulk potential to zero, the highest value of the potential is determined and subtracted from the potential profile. It should be noted that the model currently needs the numerical data because forcesworking asks for them.

3.2 FindPhiS

FindPhiS helps to define the boundaries in a numerical potential profile. It begins by finding the deviation of the potential from V_0 by adding $|V_0|$ to the potential and finding at which index $\Phi + |V_0|$ is closest to 0. This index is then saved and used to determine the actual sheath thickness.

Because the data used to determine $F_{g^*}(z)$ was only determined for $0.002m \leq z \leq 0.004m$ (as measured from the electrode), FindPhiS will also reduce the data for the potential to the extent relevant for the model. Since the model, the measurements and the simulation all use different coordinates, the coordinates have to be transformed to all comply with the model. The systems will in shorthand be called O, O' and O", respectively. How the boundaries in the different coordinates relate is shown in figure 13. Rmmprime and Lmmprime are known variables, as the range within which data for the position of the particle is available (zspan) is known in O'. From these, Rmm and Lmm can be found in O. Then, using the same method as for V_0, the indices at which the position is equal to Rmm and Lmm can be found. Using these the range of Φ can be reduced to only the extent relevant for the model. This data is then saved as a .ascii file to become available for fitting.

3.3 selectvars

Selectvars is a relatively simple script, consisting of (currently) 15 similar sections. It begins by defining an empty vector (the selectorvector) that stores the information acquired when running the script. For each used index, the script shows a few prompts for selecting options for the variables. An example is shown in figure 14. Each parameter has a 'term' assigned to it, in this example the term is 'TiA'. All matlab variables associated with this parameter will start with this term. In the example, line 19 gives TiA a position in the selectorvector, and line 20 states the number of options for TiA. Line 21 lets the person running the

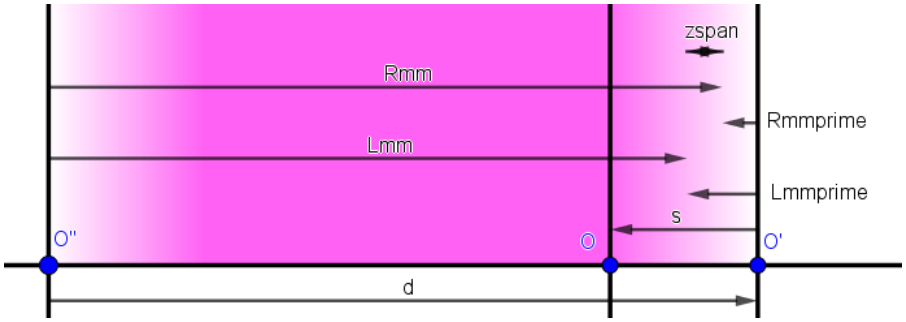


Figure 13: The extent of several distances in the different coordinate systems. O is the origin of the model, O' the origin of the measurements and O" the origin of the simulation data. zspan is the relevant range, L- and Rmm give the left and right boundaries of the relevant range in the simulation data, L- and Rmmprime the boundaries in the measurements, s is the sheath thickness and d is the distance between the electrodes.

```
18    %% Average Ion temperature       TiA
19 -  TiAX=1;
20 -  TiAN=3;
21 -  disp('Ion temperature profile: ');
22 -  TiAY=int32(input('Constant(1),Linear(2),Exponential(3): '));
23 -  while TiAY < 0 || TiAN < TiAY
24 -      disp(hint1)
25 -      TiAY=int32(input('Constant(1),Linear(2),Exponential(3): '));
26 -  end
27 -  if TiAY ~= 0
28 -      selvec(TiAX)=TiAY;
29 -  end
30
```

Figure 14: Section of the selectvars script that assigns the desired variable option to the appropriate index of the selectorvector.

program know what they will be selecting an option for, and line 22 asks for a choice. TiAY needs to be an integer between 0 and TiAN, and if it isn't, lines 23 through 26 will continue asking for new input until it is. Then, if TiAY was not set to 0 (indicating no change to the choice), the script will set the value in the selectorvector linked to TiA to the choice selected. It is possible to change one parameter choice without running the entire script, by typing 'selvec([term]X)=[choice#]'. It is not recommended to do this for all variables, though, as the shear number of combinations of options makes it easy to make mistakes. It is also probably not quicker than simply running selectvars all over again.

3.4 forcesworking

Forcesworking (versions 4 and up) defines all options for all variables, and then sets the option chosen by selectvars as the final option. It begins by defining the constants of nature and constants in the setup. After that, like selectvars, forcesworking is separated into sections for each parameter. One such section in the current version (7) is shown in figure 15 as an example. Lines 168, 170 and 172 define the different options for Phi, and the lines above them explain where these expressions were derived. Line 173 defines a cell array that contains all expressions for Phi, and line 174 sets the final Phi that is used as a variable in all expressions (Phif) to the choice made. There is also a short script, 'redefiner' that only contains all versions of line 174 for every variable; i.e. it redefines the final parameters if a new option has been chosen for any parameter. In order to be able to run redefiner, forcesworking must have been run before. Lastly, line 176 defines a string that allows easy plotting of the variable. By typing 'evalc(plot[term])' in the command window, the string can be turned into a command and will be run, so that the plot will be made.

The current working version is 7. Versions 1, 2 and 3 did not have a separate 'selectvars'. For version 4, selectvars was separated from forcesworking so that it was not necessary to fill in the entire selection list

```
166     %% Potential
167     % Child law potential
168 -   Phichild    =@(Tef,z) ((z+s0)./sf).^(4./3).*DeltaV;
169     % Numerical potential, fitted to the function C.*(A-x).^B +D
170 -   Phinumrt    =@(Tef,z) 1171.94967.*(0.03801-(z-(snum-0.04))).^0.78531-13.4034;
171     % Numerical potential, fitted to a 3rd order polynomial function
172 -   Phinumpol   =@(Tef,z) 4922.61401-436128.7157.*(z-(snum-0.04))+12879600.*(z-(snum-0
173 -   PhiV={Phichild,Phinumrt,Phinumpol};
174 -   Phif=PhiV{selvec(PhiX)};
175
176 -   plotPhi='fplot(@(z) Phif(Tef(z),z),[z1 z2])';
177
178
```

Figure 15: Section of the forcesworking7 script that defines the expression of the different parameter options, and sets the chosen option to be the final expression.

all over again to change only 1 variable. Version 5 contains more parameters and parameter options than version 4 and in particular allowed inclusion of the numerical data. Version 7 is largely the same as version 5, except all multiplications, powers and divisions were replaced by elementwise multiplications, powers and divisions. This was done to be able to plot functions of the parameters not only as function handles but as vectors. A similar attempt was made for version 6 already, but somehow that was unsuccesful. Hence 7 rather than 6 comes after 5.

3.5 plotcompare

plotcompare is a short script that makes it easy to determine the influence of one parameter on another. It asks for two parameters as input, and will then plot the latter variable for all options of the former. Give, for instance, the first variable 'TeA' and the second 'Fi'. plotcompare will then show the ion drag force for a constant, linear and exponential electron temperature profile in the same graph. It is important that forcesworking has been run before.

3.6 plotcompare2

plotcompare2 is a very different script than plotcompare. Instead of comparing 2 parameters, it automatically compares all combinations of the model to see which works best. It currently contains 15 nested 'for' loops to run over all different possible selectorvectors and assigns a quality of the model to each combination. As mentioned in subsection 2.4, the net force on the particle at every point should be equal to zero, and thus the best model is the model for which F_{net} is most equal to zero. This was determined by summing the squares of the values of F_{net} for all z. Using the square of the values ensures that all deviations are taken positive and magnified, so that a function that is 0 once but does not even remotely resemble the function $y = 0$ is considered less accurate than a function that is never 0 but is very similar in shape to $y = 0$. Suppose the graphs in figure 16 represent two outputs of the model for different parameter options. The blue line is the line $y = 0$ plus only a constant offset. The orange line, however, is the line $y = x - 2$; a line completely different than $y = 0$. Hence it is assumed that the blue line represents a better model. The sum of the deviations from zero is equal for both lines: 4.01[1]. However, the sum of the squares is 34% larger for the orange line than for the blue one. This shows the effect of squaring.

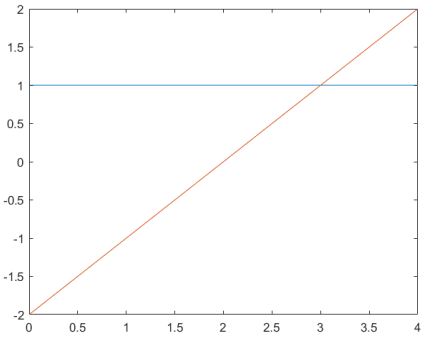


Figure 16: Two possible functions of F_{net}: $y = 1$ (blue), $y = x - 2$ (orange).

[1]It is 4.01 instead of 4.00 because matlab is measuring with a step size of 0.01 and thus in fact sums from -0.005 to 4.005.

3.7 compareanalyser

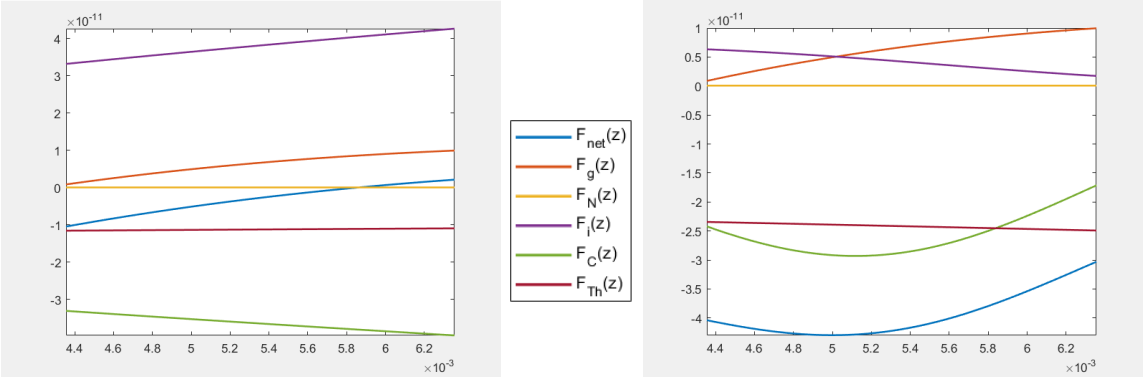
This final script, compareanalyser, determines the best combination of options out of all the combinations calculated by plotcompare2. It consists of three steps. The first step is asking what type of mininum it has to determine. Since a large part of the combinations have an imaginary part, there are different ways to calculate which value is actually smallest. The one with the smallest modulus of the numbers can be used, the smallest real part of the numbers, the smallest complex value or the smallest value of the ones that only have real parts. Unfortunately, because gravity was determined on a fixed interval, the boundaries for the model had a limited range of possibilities. In particular, the sheath has to be at least 4 mm thick, which only the numerical sheath turned out to be. Hence in this step a small transformation is made to include only the models with the numerical sheath.

The second step is to find this smallest value. With the function 'min()', the minimum value and its index can be determined easily. However, the array that contains the qualities of the different combinations is 15-dimensional, and the function 'min()' outputs only one number to index the array, such as 3045812, to indicate that the 3045812th element of the array is the smallest. More preferable is an output that states something along the lines of "The choice for TiA was 2, for TeA was 3" etcetera. Matlab does not have an automatic function for this, so a rather circuitous method containing yet again 15 nested 'for' loops is used to find this array of indices. In case there are multiple models with the same quality, all the combinations giving that output will be presented.

Once this is found, the script produces some output in step 3. It outputs the quality of the model and the choices of the variables. Then, the user will be prompted to select one of these best combinations, after which the indices are used to set the selectorvector to these choices, after which the five particle forces as well as the net force will be plotted.

	T_i	T_e	σ_i	σ_e	s	Φ	u_i	n_i
Calc index	2	1	1	1/2	5	1	2	1
Meaning	lin	const	lin		num	child	child	const
Expec index	3	3	3	2	5	3	2	2
Meaning	exp	exp	exact	poly	num	poly	child	E
	n_e	v_{T_n}	∇T	F_i	F_N	F_{Th}	F_C	
Calc index	1	1	1	2	2	1	1/2	
Meaning	const	rms	lin	cut-pol	slow	fast		
Expec index	2	1	2	3	2	2	2	
Meaning	bol	rms	exp	long-pol	slow	slow	dip	

Table 2: Combinations of the calculated and expected best model. 'lin' = linear profile or fit ; 'exp' = exponential profile or fit ; 'const' = constant profile ; 'num' = numerically derived expression ; 'child' = by Child's law ; 'E' = derived from energy conservation ; 'bol' = boltzmann distribution ; 'rms' = root mean squared ; 'cut-pol' = cut-off with poly-energetic ions ; 'long-pol' = extended potential with poly-energetic ions ; 'dip' = plus dipole term



(a) Calculated best model

(b) Expected best model

Figure 17: Plots of the 4 forces on the interval zspan.

4 Results

In this section, the results coming from running plotcompare2 and compareanalyser with the current set of parameter options will be outlined. Furthermore, some attention will be paid to how the model works.

4.1 Outcomes of the calculations

Table 2 shows the results of the calculations and the expected model. It should be noted that, although only one calculated best model is shown, all 4 methods of calculating the best model had been tried. The quality of this best model, however, was simply a real number, so that all 4 methods yielded the same result. The quality of this model was $4.674 \cdot 10^{-26}$ and plots of the forces over zspan are shown in figure 17a. Figure 17b shows the forces for the expected functions. Furthermore, there were four models with a quality of $4.674 \cdot 10^{-26}$. For these models, only σ_e and F_C changed: both could be either 1 or 2. This demonstrated that for this model it did not matter whether the linear or squared profile of the electron-neutral scattering cross section was used, nor did adding a dipole term to the coulomb force offer a significant change in the net force.

Furthermore, at the end of 2.3, the claim was made that F_N might be negligible. Hence the maximum atom drift velocity was determined to see if the magnitude of F_N in the best case scenario was much smaller than

the other forces. The maximum values of F_N for the calculated and expected models were $6.01 \cdot 10^{-14}$ and $7.61 \cdot 10^{-14}$ N, respectively. The other are on the order of magnitude of 10^{-11} N.

4.2 Quality of the model

Lastly, since part of the project to 'create a model' was writing the code of the model, the quality of the code will be discussed here. It is important to recognise that 'good code' is rarely unambiguously defined, which makes it difficult to make statements about the model without adhering to personal preferences. The shear amount of code and the fact that the scripts can better be understood by people with a background in plasma physics makes it non-feasible to do an elaborate survey to determine whether the code is good and hence only a few small points will be addressed here, as objectively as possible.

There is a lot of parallelism in the naming of variables, in particular in the scripts selectvars and forcesworking. The result is that not every variable needs to be remembered or looked up separately, only the way in which the variables are named. This parallelism is also visible in the build-up of these same scripts, due to which expansion of the model can in large part be done by copying and pasting pre-defined code to the model. This means that it is not necessary to fully understand the script to be able to expand the model, so that it becomes accessible to people that have less experience with matlab. Plotcompare2 and compareanalyser are not as user-friendly. Adapting these to accommodate a new parameter is tedious and requires meticulous care not to accidentally skip a variable. This can cause errors or flaws in the data.

The scripts are rather reliable in how they currently work. Except for a large portion of the combinations yielding non-real forces, the script does not crash and can thus give some form of output on all of the currently 5 million combinations. Calculating these, however, does take a lot of time. To obtain the results in subsection 4.1, plotcompare2 had to run for approximately 60 hours in total. Although a faster computer can probably decrease this runtime significantly, this is a long interval in which matlab can't be used for anything else either, which is impractical.

5 Conclusion

The goal of this thesis was reached in part. A program for building and testing a model was designed in the form of a series of matlab scripts, as explained in section 3. A plethora of expressions was found, outlined in section 2, and the model was tested for these expressions, the results of which can be found in section 4. The model works and is relatively organised, but it could do with some optimization.

The expected best model does not comply with the calculated best model very well. Only 5 out of 14 expressions agree between the two models, 2 of which agree because their impact on the model is negligible. Whether the model or the theory is at fault is at this moment unknown. The takeaway is thus that a model has been made, but something is still off and there is more work to be done.

6 Discussion

In this section, a brief critical discussion on what should have and could have been for this model will be presented. Subsection 6.1 will elaborate on things that (might) have gone wrong, and subsection 6.2 will present some suggestions on how the model can be expanded or improved.

6.1 Shortcomings

The first thing that is most likely to have gone wrong is a unit mistake with the electron temperature. It is common practice in plasma physics to use the electron temperature in electron volts rather than Kelvin. Although care was taken to transform T_e to K in all expressions where it was used in electron volts, a few may have slipped through the cracks. Since 1 eV is about 11600 K, interchanging the two can have serious ramifications for the expressions.

Secondly, in many sources of literature one will find that a change in e.g. pressure by a single order of magnitude can make the difference between a collisional or collisionless plasma sheath, each accompanied by their own, separate expressions. This goes to show that a plasma is highly susceptible to the conditions in which it is produced. First of all, this makes it very likely that a few of the expressions used lie within the incorrect regime of plasma parameters. Especially the intermediate regime for collisionality may prove to be more accurate, as the mean free path is not significantly smaller or significantly larger than the sheath thickness, but rather on the same order of magnitude. Secondly, because of this susceptibility one would not expect that the RF plasma can be characterized by slightly changing expressions derived for a DC plasma. Verifying the expressions for applicability to RF plasmas is thus also expected to prove fruitful.

One would expect that the sheath and the forces in it are in some way influenced by the applied RF potential. However, the model disagrees. Not a single expression includes V_{rf}. This seems odd and could thus very well just be wrong.

Whilst on the subject of potentials, it should be noted that in finding the wall potential, ΔV, quasi-neutrality was assumed up until $z = 0$ so that the assumption $\Phi(0) = 0$ could be made. As mentioned, this is simply not true. This was 'fixed' by adding the sheath-edge potential to the potential found in equation 41. However, changing the sheath-edge potential would probably influence the entire sheath potential so that the found value of ΔV would no longer be correct.

Furthermore, there's something iffy about the Bohm criterion. It was assumed that because $u_{i,0}$ had to be larger than or equal to $u_{i,B}$, the sheath starts when the two velocities were equal to one another. However, that's possibly a misconception that caused $|V_0|$ to be much smaller than it in practice is. Since the ion velocity at the sheath could already be much larger than the Bohm velocity, the potential could also be much larger, in which case the potential profile would comply much more with simulation data.

Another mistake that was probably quite impactful for the results is the fact that initially no attention was paid to the possibility that some models gave complex output. For instance, the Taylor expansion of the ion density is negative over most of the sheath, which results in a non-real Debye length, which eventually amounts to non-real forces as well. In the absence of an opportunity to re-run plotcompare2, it was decided to enable calculating the best model in several different ways, in order to allow the user to define 'the best' for themselves. In actuality, plotcompare2 should be run again in its entirety with the modification that only the real part or the modulus of the model output is used to determine the quality of the model.

Lastly, in the model it is assumed that the particle radius has the exact value 2.48 μm. However, it is known that this radius can vary quite a bit. Since gravity scales with the radius to the third power, it is very well possible that the actual force of perceived gravity is much higher or lower (just a 26% increase in the radius can double the mass and thus F_{g^*}). Hence, it is most likely very fruitful to ensure that the dust particle radius is known more accurately, or to determine $g^*(z)$ for many particles with a known average radius so that the average force of gravity can be determined. Furthermore, $g^*(z)$ was fitted to just 7 points. The reliability of the measurement can probably be increased by plotting over more values of perceived gravity.

6.2 Suggestions for further research

The first and most obvious suggestion for future research is to make the model comply with observations and numerical data. As much as the model works insofar as it generates an output, that output is not what our instruments, or even our eyes, say. For the model, this will most likely be the most pressing matter.

It might also be a good idea to test the model against numerical data first. The numerical data seem to approach the realm of possibilities more closely than the analytical expressions, and comparing the two may help to determine whether the expressions for the forces or the parameters are wrong or the expressions for the parameters.

Another suggestion to find out why the model is inaccurate is to use plotcompare to determine whether the found model is in fact the best possible model. If plotcompare shows that changing one variable gives a better model, that indicates that the method to find the best model with plotcompare2 is not the best way to go.

Since the temperature profiles are based on very vague assumptions, it is probably very fruitful to perform an in-depth analysis of the temperature profiles in order to improve them.

Havnes et al. [26] derived a very interesting expression for the thermophoretic force in close proximity to walls. However, apart from interesting it was also so complicated that matlab was unable to work with it. Hence it was not included in the thesis. However, as the model is designed to describe the plasma sheath (which is by definition close to the wall), it may still prove to be a good addition and bears checking out. The thermophoretic force could in general do with some more expressions. Initially it was thought that it had 5 different expressions, but these all turned out to be different only in notation.

Finally, only one expression for the particle charge and potential was found. This seems to imply that particle charging always happens in the same way, which is unlikely. Therefore looking for more expressions for these parameters in different regimes is likely to yield interesting results.

A Code

All the code used for the model.

A.1 Code manual

```
% Forcesworking7manual (fw7manual.m) is an empty script that provides you
% with a bunch of copy-paste terms useful for understanding and/or adapting
% forcesworking7

% You know, maybe at this point I should stop calling it fw7manual and
% actually accept that it is the manual for 5 scripts besides forcesworking
% Oh well.

% The scripts (in the order in which you should run them) are:
% loadnumerics, FindPhiS, selectvars, forcesworking7, plotcompare,
% plotcompare2, compareanalyser

% If you continued with the model and managed to fix it, I'd very much like
% to know, so please send me an email at w.b.meekes@student.tue.nl

%% OVERALL EXPLANATION OF THE SCRIPTS:

% First thing to do is run loadnumerics and then FindPhiS. Basically the
% data we have now should suffice to use as numerical data, but if you
% really want to add more numerical data, I leave that up to you. Because
% of the different ways in which you could want to add numerical data, I
% will not explain how to do that; I will simply wish you good luck.
% If you just want to use the scripts, you're in luck! All you have to do
% is run them and all the data will be loaded and named for use in the
% other scripts. Do bear in mind that at present forcesworking7 needs the
% numerical data to work properly, even if you don't use it.

% Then run selectvars.
% Upon running the script, you will be prompted to select particular
% variables out of a list of options.
% If this is the first time running the script since the last 'clear all'
% or after starting matlab, you will have to select something for every
% variable. You can do this by inputting the number corresponding to the
% variable you want. It will accept only numbers, and if the numbers are
% not integers, they will be rounded to the nearest integer. For a reliable
% script it is recommended to input only integers, but I'm not your boss so
% do whatever you want.

% Once selectvars is done, you can run forcesworking4 to define the
% variables you selected. (By 'define the variables you selected' I mean
% 'define all possible variables and confirm only the ones you selected'.
% It could be more efficient, but this was easiest for me. If you find that
% the script takes too long to run (which it shouldn't, unless you find
% several seconds 'too long'), you can change the script yourself, or buy a
% better computer or patience.)
```

```
% If after running the script you want to change one variable, you can do
% the following:
% Each variable has a 'term', which is in all variables relating to it. The
% ion sheath density has term 'nis' and the coulomb force 'FC', for
% instance. The current choices for the variables are stored in 'selvec'
% - which is short for the awesome name 'selectorvector' but it had to be
% easy to change the choices so 'selectorvector' was too long for
% convenience - with the index '[term]X'. So the ion sheath density is
% stored as 'selvec(nisX)'.
% To change the choice, (you need to have remembered the number of your
% choice) simply type 'selvec([term]X)=[your choice]' in the command
% window, then run forcesworking4 again. For instance 'selvec(TeAX)=3' to
% select an exponential electron temperature profile.
% Of course it is also possible to run selectvars again entirely.


%% EXPANDING THE SCRIPT

% To add a new variable is rather straightforward. Simply label the variable
% in such a way that you (and perhaps others) are able to recognise it. Add
% it as a function handle or constant (depending on what it is) and if you
% desire to, add a prompt for easy plotting and insert it into whichever
% function depends on it.

% To add a new option to an existing variable requires more care. Begin by
% adding the new function(s) in forcesworking4. You can do this as you
% would a new variable. Then write the line that defines the cell
% object that contains your options and the one that selects the final
% expression from that cell object. It should look like this:

% % [Description of your variable] [term] % [unit]
% % [Description of this option]
% function1 =@(vars) firstfunction;
% % [Description of this option]
% function2 =@(vars) secondfunction;
% [term]V={function1,function2};
% [term]f=[term]V{selvec([term]X)};

% For an example, look for instance at the ion sheath density nis.

% Please bear in mind that all options should be a function of the same
% variables, in the same order. If the new function depends on variables
% that the old function does not, add the variables to the variable list of
% the old function anyways. Do the same the other way around. Also, adding
% new variables means that new variables have to be inserted wherever the
% function is called from. You can use 'ctrl+H' to find and replace the
% function with its old dependences by its new dependences.

% After doing this, move over to selectvars.

% The formula below outlines the general format for adding a select menu
% for parameters with different options.
% You can easily copy-paste this after variables that depend on your new
% variable and un-comment it using ctrl+T. Then replace the variables
```

```
% N,Y,X by their equivalent for your new variable. (If the term for your
% new variable is 'abc', replace N by abcN, etc.)
% X is the index of the variable in selvec. The value does not matter, so
% long as the number is not in use by another variable. If there is no
% space left in selvec, you can simply expand it.
% N should be equal to the number of different expressions your variable
% has.
% disp(''); should contain a prompt that allows a reader to identify what
% variable they are currently selecting an option for.
% input('') should contain a prompt that makes clear the different options
% in cell object V, and gives the index of each expression in cell object
% V.

% [General name] [term]
% X=;
% N=;
% disp('');
% Y=int32(input(''));
% while Y < 0 || N < Y
% disp(hint1)
% Y=int32(input(''));
% end
% if Y ~= 0
% selectorvector(X)=Y;
% end


% To allow easy plotting, a plot-prompt is defined after each new variable
% definition:
% plot[term]='fplot(@(z) [term]f(&&&),[0 sf])';
% Replace [term] by the base name of the function you want to plot, and
% make sure to add the dependences of the variable and its dependences'
% dependences in place of the &&&.
% To plot the function, type 'evalc(plotTi);' in the command window.
% To make the Y-axis logarithmic, type in the command window:
% set(gca, 'YScale','log')

% If you get an error 'not enough/ too many input arguments' that means that
% somewhere you missed a variable dependence or accidentally added a
% variable to an expression. It is also possible that you made a mistake
% with brackets that makes matlab think you missed a variable. The best way
% to solve this is to look at the function that is giving an error, seeing
% what variables it should depend on and trying to plot the variables it
% depends on to eliminate which of them is wrong. If one of them gives an
% input argument error, the mistake is in that variable, and you do the
% same for that variable, all the way until you've found where you missed
% it.
% If none of the variables separately give an error, try re-inserting the
% expressions into the variable you're trying to plot. Probably something
% went wrong moving the dependences, so ctrl+c, ctrl+v is your friend here.

% These four scripts can also be run together by running initialrun.

%% ANALYSING YOUR DATA
```

```
% If you want to determine which model is the best, there are two scripts
% you can use: plotcompare and plotcompare2.

%% PLOTCOMPARE
% plotcompare allows you to determine the effect of changing a variable on
% another variable. You get two prompts: 'Which variable? ' and 'Variable
% to compare: '. In the first prompt you input which variable you want to
% change, and in the second you give the variable that you want to plot. It
% is important to give the input as strings.

% For instance, to see the effect that changing the electron temperature
% profile has on the ion drag force, type 'TiA' in the first prompt, and
% 'Fi' in the second. You will see three plots of the ion drag force, one
% for constant temperature, one for a linear temperature profile and one
% for an exponential temperature profile.

% plotcompare also saves your old choice for the variable and reloads it
% once it is done.

%% PLOTCOMPARE2
% plotcompare2 is in theory a very useful tool, but it's a little
% impractical. In just 60 hours, plotcompare2 can run all currently 5
% million options for the model and assign a quality value to them.

% forcesworking7 (and yes, this has to be 7) must have been run at least
% once before running plotcompare2 because it needs the data for TiAN TeAN
% etc.

% The calculated values can be stored in the file valuevec.mat, so that you
% can interrupt the program and continue later. Upon starting the script,
% matlab asks if you want to start anew (or load old values). When the
% script is finished, matlab also asks if you want to save the valuevec.
% If you want to load an old valuevec and continue from there, you can
% change the extent of the for loop. For instance, if you already
% calculated all variables up until 311111111111111 (You only need to
% calculate all variables for the exponential ion temperature), change the
% range of the for loop for the first index to i1=3:TiAN; and matlab will
% skip the constant and linear profiles.

% Basically the script is just 15 nested for loops so that it runs by all
% different combinations of parameters. Each loop stands for one parameter,
% and during each loop, the value in selvec of that parameter will be
% changed to the index of the for loop. The actual action is inside the
% 15th loop, where the selvec is applied using redefiner, after which the
% net force for these parameters will be calculated. The quality of this
% model (Fnetdev) is calculated by integrating over the deviation from
% zero squared of this function. The appropriate index in valuevec will
% then be set to this value of Fnetdev.

% At the end of the script, matlab asks you whether you want to save what
% you calculated. If you choose no, you can still save it manually.
% Bear in mind that if you save it automatically, it will be saved to
% valuevec.mat and if you already had a valuevec.mat file, that will be
% overwritten.
```

```
% Furthermore, in the script you can find currently at i8 a 'dispvec'. This
% display vector will show the progress of the script. The set 9-15 takes
% about a minute. If you want more or less output, you can move these two
% lines to another loop. Move it to i9 and you get more output, move it to
% i7 and you get less output. Simple as that.
% At i6 you find a clock item. This outputs the time every time i6 ticks.
% This is about every seven minutes. I found it very convenient to be able
% to tell accurately how long the script had already been running because
% it gives an indication of how long it still needs to run. For this one
% also, move it to another line to get more or less output.

% In the folder
% \\physstor\epg-common\Experimental_Data\Centrifuge\HyperGravityExperiments\
    ↪ ForceModel_BEPWouter\Matlab
% you can find a file named 'Dependenclists.txt', which contains the
% dependences of all variables on the variable z (input as 'zar'). You can
% copy-paste these in the indicated line below to test for other variables,
% or copy-paste them to other locations to get the full dependences of the
% function. In the latter case, do change 'zar' to 'z'.
% Please bear in mind that you can let this script output whatever variable
% you desire, but it will not make sense unless you plot Fnet. That is the
% only variable that is always supposed to be equal to zero.

% Also, a weakness that was realised only when it was too late is the fact
% that by running redefiner during the script rather than forcesworking,
% although it saves time, this way you fail to change values like s and V0
% according to the changing temperature profiles. Consequences are
% somewhere in between bad and catastrophic. So ehmm. Yeah. Quite the pity.

% The program does not have to be adapted to accommodate more options for
% the existing variables; the [term]N-values will adapt if you added the
% option correctly.
% To add a whole new variable with new options is rather simple as well. If
% your variable has term 'example', add an extra dimension to the zero
% function of valuevec (line 8), by simply writing 'exampleN' in the
% zeros() command. Separate it from the other variables with a commma.
% I'll have to admit that I'm not sure if the order matters a whole lot,
% but just to be sure I would recommend keeping the order logical. Add new
% variables always to the end of the valuevec and keep them the newest
% variable in the nested for loops.
% Also add the variable to the for loops (preferably at the end).
% Directly below the last line, type
% for i16=1:exampleN; selvec(exampleX)=i16;
% And add an extra end to the series of ends, separating them by a
% semicolon.
% Lastly, make sure to let matlab select the right index for valuevec to
% set to Fnetdev, by also adding i16 to the last line inside the for loop.

%% COMPAREANALYSER

% compareanalyser allows you to extract data from valuevec. As of this
% moment, Fnetdev can be non-real, and thus so can the values in valuevec.
% You will be prompted to select a way of calculating the minimum of the
```

```
% vector. If you changed plotcompare2 to only allow real values, you can
% remove this portion of the script.

% Currently, all sheaths except the numerically calculated one result in
% non-real values, so the final array it will determine the minimum for
% only the numerical sheath thickness.

% Then the minimum will be calculated, as well as all the combinations of
% parameters that result in this minimum.

% Then the fun begins. Matlab outputs the positions of the minima in the
% total index of the minimum in the array. So rather than outputting
% 2,3,4,1,2,3,1,2,3,1,2,3,1,2,3 (or something), it says 314823. Although
% both expressions point at the same number, the latter does not give
% information about the model that resulted in that number, whereas the
% former does. So, in come the 15 nested for loops.
% The number 234 = 200 + 30 + 4 = 2*10*10 + 3*10 + 4
% This is simply because in our decimal system, all digits can run up to
% 10. 234 indicates 4 counts + 3 times 10 counts + 2 times 10 times 10 counts.
% But what if the first digit can only go to 3, the 2nd to 4 and the 3rd to 5?
% Then 234 indicates 4 counts + 3 times 5 counts + 2 times 4 times 5 counts
% = 59.
% The same goes for this array, where the maximum number corresponding to
% each index is [term]N. This is how indexval is calculated. For every
% combination of indices, it calculates the final number it should have in
% the array.
% Once the indexval corresponds to the index of the minimum that is
% currently regarded, it will set the indices to the correct value in
% minarr.

% At the end, the results will be shown and you will be prompted to select
% one of the results for plotting.

% This script should also adapt automatically to new options for existing
% variables.
% If you add an entirely new variable, make sure to add a new nesting to
% the for loops, analogously to plotcompare2.
% Also add the index id16 to posvec, preferably at the end, separated from
% the other indices.
% Then make sure indexval is adapted to the new parameter. At the end of
% indexval, add the product of all N-values that came before exampleN, and
% multiply that by (id16-1). For example, sigin is the third in the array,
% TiA and TeA came before it, so the sigin contribution to indexval is
% equal to +TiAN*TeAN*(id3-1).
% The -1 is to account for the fact that the indices start counting at 1.
% Lastly, don't forget to set selvec to the appropriate value with the line
% selvec(exampleX)=minarr(option,16);
% At the end of the script.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Some specific comments:
```

```
% up
% The particle velocity is a bit obscure. It is assumed negligible, in part
% because it is very difficult to find an expression for it.

% Tic
% In order to ensure that the constant temperature profile can be called
% in the same way as the actual profiles for Ti (and Te), Tic will here be
% transformed to a function handle.

% V0s
% This sheath-edge potential is used to determine the sheath thickness,
% which is necessary for the decaying temperature profiles. Since the
% sheath thickness and sheath-edge potential depend on each other, a simple
% expression needs to be derived before being able to use options that
% require the sheath thickness.

% FacHavint
% This is more of a placeholder in case a broken thing should magically
% find itself fixed some day.
% Basically the idea was to find the expression for the factor that Havnes
% proposed to add to the thermophoretic force to take wall effects into
% account (which includes an integral that returns the meijerG function
% that matlab appears to have trouble plotting) by numerically integrating
% the integrand using a loop. Didn't work even a little, so a new plan is
% underway. Maybe.

% FacHavY
% Also a placeholder

% hint1
% This is a string that is prompted when a mistake is made selecting
% variables.
```

A.2 Load simulation data

```
% loadnumerics Loads data to the workspace for plotting Ion and Electron
% densities and the potential.

% Load meshes for R and Z
load RPlaneMesh.mat
load ZPlaneMesh.mat
% Load the potential in tempdata and name it something useful
load AveragedPotential.mat
Phinum=TempData;
% Load electron density in tempdata and name it something useful
load AveragedElectron.mat
nesnum=TempData;
% Same for ions
load AveragedIon.mat
nisnum=TempData;
```

```
clear TempData;

% Set bulk potential (highest potential) to zero
Phimax=max(max(Phinum));
Phinorm=Phinum-Phimax;

% Text for plotting: (replace TempData by what you want to plot
% surf(totalRPlane,totalZPlane,TempData);shading interp

% This prompt can be used to plot Phi on r=0.
plotPhinorm="plot(totalZPlane(:,1),Phinorm(:,1));";
```

A.3 Determine Φ and s

```
% FindPhiS determines Phi and s from numerical results
% You need to run loadnumerics first.

% Define the bohm criterion
Teb=29005;
bohmV0=4.3086652*10^-5*Teb;
% Find the value of the sheath-edge potential in phinorm
% First find the deviation from bohmV0
Phired=abs(Phinorm(:,1)+bohmV0);
% Then find the number with the smallest deviation from bohmV0
% (The upper half is used because we want the thickness of the sheath at
% the right electrode.)
% Dimensions of the Phinorm array (so we know what 'half' is).
[hor,vert]=size(Phinorm);
% using 'hor/2' might give trouble if hor is not an even number, warning:
if mod(hor,2)==1
    disp('Horizontal size of array is odd, expect trouble.')
end
[Phisf,sindex]=min(Phired(hor/2:hor));
% Add the found index to half the Phi indices
sindex=hor/2+sindex-1;
% Convert to actual distance
snum=0.04-totalZPlane(sindex,1);

% Since experimental data is only available within the range of 2-4 mm from
% the rf electrode, those bounds will be set to determine what the
% potential profile will look like.
% mmprime gives the position of the boundaries in plasimo's coordinate
% system, where the electrode is at z=0.
Lmmprime=0.004;
Rmmprime=0.002;
Lmm=0.04-Lmmprime;
Rmm=0.04-Rmmprime;
% Find the deviation of totalZPlane (the vector that indicates what index
% in Phi corresponds to what distance) from the bounds
zdevL=abs(totalZPlane(:,1)-Lmm);
zdevR=abs(totalZPlane(:,1)-Rmm);
```

```
% Minimise both
[Ldev,Lbound]=min(zdevL);
[Rdev,Rbound]=min(zdevR);

% The relevant data for Phi (which will be used for fitting) is then
relevantZ=totalZPlane(Lbound:Rbound,1);
relevantPhi=Phinorm(Lbound:Rbound,1);
relevantvars=[relevantZ relevantPhi];

save('varsPhiplot.txt','relevantvars','-ascii');
```

A.4 Choose parameter options

```
% selectvars provides a script that allows you to select new variables for
% forcesworking4. This feature was not implemented in forcesworking3. It
% works with forcesworking 4 through 7.

selvec=zeros(20,1);

%$#@!
hint1='Please input one of the following numbers or 0:';

%% Average Ion temperature TiA
TiAX=1;
TiAN=3;
disp('Ion temperature profile: ');
TiAY=int32(input('Constant(1),Linear(2),Exponential(3): '));
while TiAY < 0 || TiAN < TiAY
    disp(hint1)
    TiAY=int32(input('Constant(1),Linear(2),Exponential(3): '));
end
if TiAY ~= 0
    selvec(TiAX)=TiAY;
end

%% Average electron temperature TeA
TeAX=2;
TeAN=3;
disp('Electron temperature profile: ');
TeAY=int32(input('Constant(1),Linear(2),Exponential(3): '));
while TeAY < 0 || TeAN < TeAY
    disp(hint1)
    TeAY=int32(input('Constant(1),Linear(2),Exponential(3): '));
end
if TeAY ~= 0
    selvec(TeAX)=TeAY;
end

%% Electron-neutral scattering cross section sigen
sigenX=16;
sigenN=2;
```

```
disp('Electron-neutral scattering cross section: ');
sigenY=int32(input('Linear(1), Squared(2): '));
while sigenY < 0 || sigenN < sigenY
    disp(hint1)
    sigenY=int32(input('Linear(1), Squared(2): '));
end
if sigenY ~= 0
    selvec(sigenX)=sigenY;
end


%% Ion-neutral scattering cross section sigin
siginX=3;
siginN=3;
disp('Ion-neutral scattering cross section: ');
siginY=int32(input('Constant(1), Variable(2),Exact(3): '));
while siginY < 0 || siginN < siginY
    disp(hint1)
    siginY=int32(input('Constant(1), Variable(2),Exact(3): '));
end
if siginY ~= 0
    selvec(siginX)=siginY;
end

%% Sheath thickness s
sX=4;
sN=5;
disp('Sheath thickness: ');
sY=int32(input('Collisionless(1),Coll w/ c lmf(2),Coll w/ c mob (3),Expected(4),numerical
    ↪ (5): '));
while sY < 0 || sN < sY
    disp(hint1)
    sY=int32(input('Collisionless(1),Coll w/ c lmf(2),Coll w/ c mob (3),Expected(4),
        ↪ numerical(5): '));
end
if sY ~= 0
    selvec(sX)=sY;
end

%% Potential Phi
PhiX=15;
PhiN=3;
disp('Sheath potential profile: ');
PhiY=int32(input('Child(1), Num root(2), Num poly(3): '));
while PhiY < 0 || PhiN < PhiY
    disp(hint1)
    PhiY=int32(input('Child(1), Num root(2),Num poly(3): '));
end
if PhiY ~= 0
    selvec(PhiX)=PhiY;
end

%% Ion velocity ui
uiX=5;
```

```
uiN=2;
disp('Ion sheath velocity: ');
uiY=int32(input('Bohm(1), Child(2): '));
while uiY < 0 || uiN < uiY
    disp(hint1)
    uiY=int32(input('Bohm(1), Child(2): '));
end
if uiY ~= 0
    selvec(uiX)=uiY;
end

%% Ion density profile nis
nisX=6;
nisN=4;
disp('Ion density profile: ');
nisY=int32(input('Constant(1),Energy(2),Taylor(3),Child(4): '));
while nisY < 0 || nisN < nisY
    disp(hint1)
    nisY=int32(input('Constant(1),Energy(2),Taylor(3),Child(4): '));
end
if nisY ~= 0
    selvec(nisX)=nisY;
end

%% Electron density profile nes
nesX=7;
nesN=3;
disp('Electron density profile: ');
nesY=int32(input('Constant(1),Boltzmann(2),Taylor(3): '));
while nesY < 0 || nesN < nesY
    disp(hint1)
    nesY=int32(input('Constant(1),Boltzmann(2),Taylor(3): '));
end
if nesY ~= 0
    selvec(nesX)=nesY;
end

%% Gas thermal velocity vTn
vTnX=8;
vTnN=2;
disp('Gas thermal velocity: ');
vTnY=int32(input('Root mean square(1), Mean(2): '));
while vTnY < 0 || vTnN < vTnY
    disp(hint1)
    vTnY=int32(input('Root mean square(1), Mean(2): '));
end
if vTnY ~= 0
    selvec(vTnX)=vTnY;
end

%% Temperature gradient gradT
gradTX=9;
gradTN=2;
disp('Temperature gradient: ');
```

```
gradTY=int32(input('Constant(1),Exponential(2): '));
while gradTY < 0 || gradTN < gradTY
    disp(hint1)
    gradTY=int32(input('Constant(1),Exponential(2): '));
end
if gradTY ~= 0
    selvec(gradTX)=gradTY;
end

% %% Sheath edge potential V0
% V0X=10;
% V0N=3;
% disp('Sheath-edge potential: ');
% V0Y=int32(input('Simple(1),Collisionless(2),Collisional(3): '));
% while V0Y < 0 || V0N < V0Y
% disp(hint1)
% V0Y=int32(input('Simple(1),Collisionless(2),Collisional(3): '));
% end
% if V0Y ~= 0
% selvec(V0X)=V0Y;
% end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Ion drag force Fi
FiX=11;
FiN=4;
disp('Ion drag force: ');
FiY=int32(input('Cutoff+mono(1),Cutoff+poly(2),Long+poly(3),+CoulombLog(4): '));
while FiY < 0 || FiN < FiY
    disp(hint1)
    FiY=int32(input('Cutoff+mono(1),Cutoff+poly(2),Long+poly(3),+CoulombLog(4): '));
end
if FiY ~= 0
    selvec(FiX)=FiY;
end

%% Neutral drag force FN
FNX=12;
FNN=4;
disp('Neutral drag force: ');
FNY=int32(input('Fast+rel(1),Slow+rel(2),HardSphere(3),Slow(4): '));
while FNY < 0 || FNN < FNY
    disp(hint1)
    FNY=int32(input('Fast+rel(1),Slow+rel(2),HardSphere(3),Slow(4): '));
end
if FNY ~= 0
    selvec(FNX)=FNY;
end

%% Thermophoretic force FTh
FThX=13;
FThN=2;
disp('Thermophoretic force: ');
```

```
FThY=int32(input('Aussems(1),Paeva(2): '));
while FThY < 0 || FThN < FThY
    disp(hint1)
    FThY=int32(input('Aussems(1),Paeva(2): '));
end
if FThY ~= 0
    selvec(FThX)=FThY;
end

%% Coulomb force FC
FCX=14;
FCN=2;
disp('Coulomb force: ');
FCY=int32(input('Regular(1),+Dipole(2): '));
while FCY < 0 || FCN < FCY
    disp(hint1)
    FCY=int32(input('Regular(1),+Dipole(2): '));
end
if FCY ~= 0
    selvec(FCX)=FCY;
end
```

A.5 Define parameters

```
% Forcesworking5 defines possible variables and forces and allows easy
% plotting of the variables and forces.
% First run selectvars

% A matlab script 'fw4manual' should be available that outlines the
% functions of the script in detail. (Yes, that is the manual for
% forcesworking4 but fw5 works alsmost exactly the same.)

%$#@! Will be displayed if extra information about a particular object is
% available in the manual.

%% Constants
% Constants of nature:
kb=1.38064852.*10.^(-23); % J./K Boltzmann constant
el=1.60217662.*10.^(-19); % C Electron charge
eps0=8.854187817.*10.^-12; % C./(V m) Vacuum permittivity

% Constants in the setup:
rhop=2160; % kg./m.^3 Particle mass density
rp=2.48.*10.^-6; % m Particle radius
nurf=13.56.*10.^6; % Hz Radio frequency
Vrf=80; % V RF voltage
kT=0.017; % W./m./K Thermal conductivity argon
p=15; % Pa Pressure
Tb=300; % K Bulk temperature
Tel=500; % K Electrode temperature
Teb=29005; % K Bulk electron temperature
```

```
ng=p./kb./Tb; % m.^-3 Gas density
ni0=3.162.*10.^14; % m.^-3 Bulk ion density
ne0=3.162.*10.^14; % m.^-3 Bulk electron density
mi=39.948.*1.66053904.*10.^-27; % kg Ion (argon) mass
me=9.109.*10.^-31; % kg Electron mass
mn=mi+me; % kg Neutral atom (argon) mass
%$#@!
up=0; % m./s Particle velocity

%z interval
znum=10.^-4; % Number of data points in z
zstep=(-Rmmprime+Lmmprime).*znum;% Calculates what the step size for z needs to be
z1=snum-Lmmprime; % Left bound
z2=snum-Rmmprime; % Right bound
zar=z1:zstep:z2; % Defines a z interval


%% Temperatures
% Wherever the neutral and ion temperatures are assumed constant, they will
% be assumed to be equal to their bulk temperatures
% Bulk Temperatures % K
% Neutrals
Tnc =Tb;
% Ions
Tic =Tb;

% Average temperatures
% Ion temperature TiA % K
% TiAX=1
% Constant
TiAc =Tic;
% Linear
TiAl =(Tel+Tic)./2;
% Exponential
TiAe =(Tel-Tb)./log(Tel./Tb);
TiAV={TiAc,TiAl,TiAe};
TiAf=TiAV{selvec(TiAX)};


% Electron temperature TeA % K
% Constant
TeAc =Teb;
% Linear
TeAl =Teb.*TiAl./Tic;
% Exponential
TeAe =Teb.*(Tel-Tb)./Tb./log(Tel./Tb);
TeAV={TeAc,TeAl,TeAe};
TeAf=TeAV{selvec(TeAX)};


%% Cross sections
% Linear fit to subramanian sigen % m.^2
sigenl =@(Tef) -8.49694.*10.^-21+1.77793.*10.^-24.*Tef;
% 2nd-order polynomial fit to subramanian
```

```
sigenp =@(Tef) -3.80964.*10.^-21+1.53036.*10.^-24.*Tef+2.30868.*10.^-30.*Tef.^2;
sigenV={sigenl,sigenp};
sigenf=sigenV{selvec(sigenX)};

% Ion-neutral scattering cross section sigin % m.^2
% Constant cross section, stated by Paeva
siginc =@(Tif) 80.*10.^-20;
% Taylor expansion of expression by Phelps 1994 around 300 K
siginv =@(Tif) (2.32022.*10.^-18) - (6.076.*10.^-17).*(kb.*Tif./el)+7.04191.*10.^-16.*(
    ↪ kb.*Tif./el).^2;
% Expression by Phelps 1994
siginex =@(Tif) 2.*10.^-19./(kb.*Tif./el).^0.5./(1+kb.*Tif./el)+3.*10.^-19.*kb.*Tif./el.
    ↪ *(1+kb.*Tif./el./3).^-2.3;
siginV={siginc,siginv,siginex};
siginf=siginV{selvec(siginX)};


%% Mean free paths
% Electron mean free path lmfe % m
lmfef0 =kb.*TeAf./p./sigenf(TeAf);
% Position dependent lmfe
lmfef =@(Tef,sigenf) kb.*Tef./p./sigenf;
% Ion mean free path
lmfif0 =kb.*TiAf./p./siginf(TiAf);
% Position dependent lmfi
lmfif =@(Tif,siginf) kb.*Tif./p./siginf;

%% More temperatures
%$#@!
% Constant ion temperature profile
Tic =@(z) Tic;

% Constant electron temperature profile
Tec =@(z) Teb;


%% Sheath-presheath potential
%$#@!
V0s=-0.5.*kb.*TeAf./el;
% Plasma-wall potential difference DeltaV % V
DeltaV=-5.2.*kb.*TeAf./el;


%% Sheath thickness
% Collisionless sheath s % m
sfree =83244.7.*abs(DeltaV).^0.75./ne0.^0.5./TeAf.^0.25;
% Collisional sheath with mean free path assumed constant
scoll =sqrt(50./27).*9388.2.*(abs(DeltaV).^3./ne0.^2./ng./siginf(TiAf)./TeAf).^0.2;
% Collsiional sheath with ion mobility assumed constant
scmob =8968.95.*(DeltaV.^2./(ng.*ne0.*siginf(TiAf).*TeAf)).^(1./3);
% Intermediate region (logarithmic average of the free and collisional
% sheaths)
% sinter =(sfree.*scoll.*scmob).^(1./(sN-1));
% Expected sheath thickness; 8 mm
```

```
sexp =0.008;
sV={sfree,scoll,scmob,sexp,snum};
sf=sV{selvec(sX)};

% Pre-sheath thickness s0 % m
s0 =1.359.*lmfif0;

%% Varying temperatures
% Ion temperature profile
% Linearly decreasing from the electrode
Tilin =@(z) Tb+(Tel-Tb).*z./sf;
% Exponentially decaying from the electrode
Tiexp =@(z) Tb.*(Tel./Tb).^(z./sf);
% 1./(1+1./z) ??
TiV={Tic,Tilin,Tiexp};
Tif=TiV{selvec(TiAX)};

%$#@!
plotTi='fplot(@(z) Tif(z),[z1 z2]);';

% Electron temperature profile
% Linearly decreasing from the electrode
Telin =@(z) Teb./Tb.*(Tb+(Tel-Tb).*z./sf);
% Exponentially decaying from the electrode
Teexp =@(z) Teb.*(Tel./Tb).^(z./sf);
TeV={Tec,Telin,Teexp};
Tef=TeV{selvec(TeAX)};

plotTe='fplot(@(z) Tef(z),[z1 z2]);';

%% Potential
% Child law potential
Phichild =@(Tef,z) ((z+s0)./sf).^(4./3).*DeltaV;
% Numerical potential, fitted to the function C.*(A-x).^B +D
Phinumrt =@(Tef,z) 1171.94967.*(0.03801-(z-(snum-0.04))).^0.78531-13.4034;
% Numerical potential, fitted to a 3rd order polynomial function
Phinumpol =@(Tef,z) 4922.61401-436128.7157.*(z-(snum-0.04))+12879600.*(z-(snum-0.04)).
    ↪ ^2-126864000.*(z-(snum-0.04)).^3;
PhiV={Phichild,Phinumrt,Phinumpol};
Phif=PhiV{selvec(PhiX)};

plotPhi='fplot(@(z) Phif(Tef(z),z),[z1 z2])';


%% Velocities
% Bohm velocity (constant) ui % m./s
uibohm =sqrt(kb.*Teb./mi);
uibohmf =@(Phif) uibohm;
% Child law sheath velocity
uichild =@(Phif) sqrt(-2.*el./mi.*Phif);
uiV={uibohmf,uichild};
uif=uiV{selvec(uiX)};

plotui='fplot(@(z) uif(Phif(Tef(z),z)),[z1 z2])';
```

```
% Different profile for an 'effective ion velocity'
uieff =@(uif,Tif) (uif.^2+8.*kb.*Tif./pi./mi).^(1./2);

plotuieff='fplot(@(z) uieff(uif(Phif(Tef(z),z)),Tif(z)),[z1 z2])';

% Gas bulk velocity % m./s
unf =0.053466;

%% Densities
% Ion sheath density, nis % m.^-3
% Constant ion density
nisc =@(Phif,Tef,z,lmfif) ni0;
% Boltzmann distributed ions
nisbol =@(Phif,Tef,z,lmfif) ni0.*(1-el.*Phif./(0.5.*mi.*uibohm.^2)).^(-0.5);
% Taylor expansion of Boltzmann distribution
nistay =@(Phif,Tef,z,lmfif) ni0.*(1+el.*Phif./(mi.*uibohm.^2));
% Child law distributed ions
nischild =@(Phif,Tef,z,lmfif) -4.*eps0./el./3.*(Phif-V0s)./(z+1.359.*lmfif).^2;
nisV={nisc,nisbol,nistay,nischild};
nisf=nisV{selvec(nisX)};

plotnis='fplot(@(z) nisf(Phif(Tef(z),z),Tef(z),z),[z1 z2])';

% Electron sheath density, nes % m.^-3
% Constant electron density
nesc =@(Phif,Tef) ne0;
% Boltzmann distributed electrons
nesbol =@(Phif,Tef) ne0.*exp(el.*Phif./kb./Tef);
% Taylor expansion of Boltzmann distribution
nestay =@(Phif,Tef) ne0.*(1+el.*Phif./kb./Tef);
nesV={nesc,nesbol,nestay};
nesf=nesV{selvec(nesX)};

plotnes='fplot(@(z) nesf(Phif(Tef(z),z),Tef(z)),[z1 z2])';

%% Particle properties
% Particle potential % V
Vpf =@(Tef,Tif,nesf,nisf) kb./el.*Tif-Tef.*lambertw(exp(kb.*Tif./el./Tef).*kb.*Tif./el./
    ↪ Tef.*sqrt(mi./me.*(nesf./nisf).^2.*Tef./Tif));
plotVp='fplot(@(z) Vpf(Tef(z),Tif(z),nesf(Phif(Tef(z),z),Tef(z)),nisf(Phif(Tef(z),z),Tef(
    ↪ z),z)),[z1 z2])';
% Particle charge
Qpf =@(Vpf) 4.*pi.*eps0.*rp.*Vpf;
plotQp='fplot(@(z) Qpf(Vpf(Tef(z),Tif(z),nesf(Phif(Tef(z),z),Tef(z)),nisf(Phif(Tef(z),z),
    ↪ Tef(z),z))),[z1 z2])';

%% Gas thermal velocity
% Root mean square of the velocity vTn % m./s
vTnrms =@(Tif) sqrt(kb.*Tif./mn);
% Mean of the magnitude of the velocity
vTnmean =@(Tif) sqrt(2.*kb.*Tif./pi./mn);
vTnV={vTnrms,vTnmean};
vTnf=vTnV{selvec(vTnX)};
```

```
plotvTn='fplot(@(z) vTnf(Tif(z)),[z1 z2])';

%% Temperature gradient
% Constant gradient gradT % K./m
gradTc =@(z) (Tel-Tb)./sf;
% Gradient of exponential decay
gradTexp =@(z) log(Tel./Tb).*Tb./sf.*(Tel./Tb).^(z./sf);
gradTV={gradTc,gradTexp};
gradTf=gradTV{selvec(gradTX)};

plotgradT='fplot(@(z) gradTf(z),[z1 z2])';

%% Impact parameters
% Critical impact parameter % m
bcf =@(Vpf,uif) rp.*(-el.*Vpf)./(mi.*uif.^2);
plotbc='fplot(@(z) bcf(Vpf(Tef(z),Tif(z),nesf(Phif(Tef(z),z),Tef(z)),nisf(Phif(Tef(z),z),
    ↪ Tef(z),z)),uif(Phif(Tef(z),z))),[z1 z2])';
% Impact parameter for collect % m
bcollf =@(Vpf,uif) rp.*sqrt(1-2.*el.*Vpf./(mi.*uif.^2));
plotbcoll='fplot(@(z) bcollf(Vpf(Tef(z),Tif(z),nesf(Phif(Tef(z),z),Tef(z)),nisf(Phif(Tef(
    ↪ z),z),Tef(z),z)),uif(Phif(Tef(z),z))),[z1 z2])';
% Smallest impact parameter % m
b0f =@(Qpf,uieff) abs(el.*Qpf./(2.*pi.*eps0.*mi.*uieff));
plotb0='fplot(@(z) b0f(Qpf(Vpf(Tef(z),Tif(z),nesf(Phif(Tef(z),z),Tef(z)),nisf(Phif(Tef(z)
    ↪ ,z),Tef(z),z))),uieff(uif(Phif(Tef(z),z)),Tif(z))),[z1 z2])';

%% Debye lengths
% Electron debye length lD % m
lDef =@(Tef,nesf) sqrt(eps0.*kb.*Tef./el.^2./nesf);
plotlDe='fplot(@(z) lDef(Tef(z),nesf(Phif(Tef(z),z),Tef(z))),[z1 z2])';
% Ion debye length
lDif =@(Tif,nisf) sqrt(eps0.*kb.*Tif./el.^2./nisf);
plotlDi='fplot(@(z) lDif(Tif(z),nisf(Phif(Tef(z),z),Tef(z),z)),[z1 z2])';
% Total debye length
lDf =@(lDef,lDif) lDef.*lDif./sqrt(lDef.^2+lDif.^2);
plotlD='fplot(@(z) lDf(lDef(Tef(z),nesf(Phif(Tef(z),z),Tef(z))),lDif(Tif(z),nisf(Phif(Tef
    ↪ (z),z),Tef(z),z))),[z1 z2])';

% % Presheath-sheath potential V0 % V
% % Here you can select a more accurate expression for V0, if need be.
% % Collisionless
% V0less =@(sf,lDf) (0.9.*sqrt(3).*sf./lDf).^(4./3).*Teb./2;
% % Collisional
% V0coll =@(sf,lmfif) TeAf.*(1./2-sf./lmfif-kb./2./el.*lambertw(el./kb.*exp(el./kb.*(1-2.
    ↪ *sf./lmfif))));
% V0V={V0s,V0less,V0coll};
% V0f=V0V{selvec(V0X)};


%% Electric field
% Child law electric field
Echild =@(Phif,z) -4./3.*(DeltaV-V0s)./sf.^(4/3).*z.^(1/3);
% Derivative of the numerical expression
```

```
Enumrt =@(Phif,z) 920.344.*(-z-0.00199+snum).^(-0.21469);
% Derivative of the numerical expression
Enumpol =@(Phif,z) 436129-25759200.*(0.04-snum+z)+380592000.*(0.04-snum+z).^2;
EV={Echild,Enumrt,Enumpol};
Ef=EV{selvec(PhiX)};

plotE='fplot(@(z) Ef(Phif(Tef(z),z),z),[z1 z2])';

%% Coulomb logarithm
% LOG % []
LOGlieb =@(lDf,b0f) 2.*lDf./b0f;
LOGN=1;
LOGV={LOGlieb};
LOGf=LOGV{1};

plotLOG='fplot(@(z) LOGf(lDf(lDef(Tef(z),nesf(Phif(Tef(z),z),Tef(z))),lDif(Tif(z),nisf(
    ↪ Phif(Tef(z),z),Tef(z),z))),b0f(Qpf(Vpf(Tef(z),Tif(z),nesf(Phif(Tef(z),z),Tef(z)),
    ↪ nisf(Phif(Tef(z),z),Tef(z),z))),uieff(uif(Phif(Tef(z),z)),Tif(z)))),[z1 z2])';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% FORCES
% N
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Gravity
% Fg
% Find a way to get the data from gravity into the form F_g(z)
% Maybe the above worked
Ag=8.54264;
Bg=0.36617;
Cg=1.87712./1000;
Fgf =@(z) (-((sf-z)./Cg).^(1./Bg)+Ag).*4./3.*pi.*rhop.*rp.^3.*9.81;

plotFg='fplot(@(z) Fgf(z),[z1 z2]);';


%% Ion drag
% Fi
% Paeva: Cut-off coulomb potential and monoenergetic ions
Fipaevacm =@(nisf,uif,lDf,uieff,bcf,bcollf,b0f,Vpf,LOGf) nisf.*mi.*uieff.*uif.*(pi.*
    ↪ bcollf.^2+2.*pi.*bcf.^2.*log((lDf.^2+bcf.^2)./(bcollf.^2+bcf.^2)));
% Paeva: Cut-off coulomb potential and non-monoenergetic ions
Fipaevacp =@(nisf,uif,lDf,uieff,bcf,bcollf,b0f,Vpf,LOGf) nisf.*mi.*uieff.*uif.*(pi.*
    ↪ bcollf.^2+2.*pi.*bcf.^2.*log(1+lDf.^2./bcf.^2));
% Paeva: Extended coulomb potential and non-monoenergetic ions
Fipaevalp =@(nisf,uif,lDf,uieff,bcf,bcollf,b0f,Vpf,LOGf) nisf.*mi.*uieff.*uif.*(pi.*
    ↪ bcollf.^2+2.*pi.*bcf.^2.*log((lDf+bcf)./(rp+bcf)));
% Fortov: For very fast ions only
% Fifort3=@ionfort3; % Defined by Fortov, beta =< 5
% Fifort4=@ionfort4; % Defined by Fortov, beta => betacr
% Liebermann: Takes coulomb logarithm into account
Filieb =@(nisf,uif,lDf,uieff,bcf,bcollf,b0f,Vpf,LOGf) pi.*mi.*nisf.*uif.*uieff.*(bcollf.
    ↪ ^2+b0f.^2.*log(LOGf));
% Fiauss=@ionauss; % Defined by Aussems
```

```
% Probably overly complicated
% Khrapak: Applied to subthermal ion flow
% FiBC=@ionBC; % Using the Binary Collision model
FiV={Fipaevacm,Fipaevacp,Fipaevalp,Filieb};
Fif=FiV{selvec(FiX)};

plotFi='fplot(@(z) Fif(nisf(Phif(Tef(z),z),Tef(z),z,lmfif(Tif(z),siginf(Tif(z)))),uif(
    ↪ Phif(Tef(z),z)),lDf(lDef(Tef(z),nesf(Phif(Tef(z),z),Tef(z))),lDif(Tif(z),nisf(Phif
    ↪ (Tef(z),z),Tef(z),z,lmfif(Tif(z),siginf(Tif(z))))),uieff(uif(Phif(Tef(z),z)),Tif(
    ↪ z)),bcf(Vpf(Tef(z),Tif(z),nesf(Phif(Tef(z),z),Tef(z)),nisf(Phif(Tef(z),z),Tef(z),z
    ↪ ,lmfif(Tif(z),siginf(Tif(z)))),uif(Phif(Tef(z),z))),bcollf(Vpf(Tef(z),Tif(z),nesf
    ↪ (Phif(Tef(z),z),Tef(z)),nisf(Phif(Tef(z),z),Tef(z),z,lmfif(Tif(z),siginf(Tif(z))))
    ↪ ),Tif(z)),b0f(Qpf(Vpf(Tef(z),Tif(z),nesf(Phif(Tef(z),z),Tef(z)),nisf(Phif(Tef(z),z
    ↪ ),Tef(z),z,lmfif(Tif(z),siginf(Tif(z))))),uieff(uif(Phif(Tef(z),z)),Tif(z))),Vpf(
    ↪ Tef(z),Tif(z),nesf(Phif(Tef(z),z),Tef(z)),nisf(Phif(Tef(z),z),Tef(z),z,lmfif(Tif(z
    ↪ ),siginf(Tif(z)))),LOGf(lDf(lDef(Tef(z),nesf(Phif(Tef(z),z),Tef(z))),lDif(Tif(z),
    ↪ nisf(Phif(Tef(z),z),Tef(z),z,lmfif(Tif(z),siginf(Tif(z))))),b0f(Qpf(Vpf(Tef(z),
    ↪ Tif(z),nesf(Phif(Tef(z),z),Tef(z)),nisf(Phif(Tef(z),z),Tef(z),z,lmfif(Tif(z),
    ↪ siginf(Tif(z))))),uieff(uif(Phif(Tef(z),z)),Tif(z)))),[z1 z2])';

%% Neutral drag
% FN
% Bouchoule 1: |up-un| >> vTn (Fast particle)
FNbouchf =@(up,vTnf,Tif) -pi.*rp.^2.*mn.*ng.*(up-unf).*abs(up-unf);
% Bouchoule 2: |up-un| << vTn (Slow particle)
FNbouchs =@(up,vTnf,Tif) -pi.*rp.^2.*mn.*ng.*(up-unf).*vTnf;
% Liebermann: Hard sphere approximation
FNlieb =@(up,vTnf,Tif) -mn.*ng.*pi.*rp.^2.*vTnf.*unf;
% Fortov: |up-un| << vTn (Slow particle)
FNForts =@(up,vTnf,Tif) -8./3.*sqrt(2.*pi).*rp.^2.*mn.*ng.*kb.*Tif./el.*(up-unf)./vTnf;
FNV={FNbouchf,FNbouchs,FNlieb,FNForts};
FNf=FNV{selvec(FNX)};

plotFN='fplot(@(z) FNf(up,vTnf(Tif(z)),Tif(z)),[z1 z2])';

%$#@!
% (FacHavint)


%% Thermophoretic
% FTh
% Fortov: Very fast ions BROKEN
% FThFort =@(gradTf,vTnf) -16./9.*rp.^2./siginf.*gradTf;
% Aussems: Probs for fast ions too; according to Fortov identical to FThFort
FThAuss =@(gradTf,vTnf) -8.*sqrt(2.*pi)./15.*rp.^2./vTnf.*kT.*gradTf;
% Paeva: ???
FThPaeva =@(gradTf,vTnf) -32./15.*rp.^2./vTnf.*kT.*gradTf;
% Havnes: Factor for near walls BROKEN
% FacHav =@(z,lmfif) 1+(z./lmfif).^6./(256.*sqrt(pi)).*(5.*meijerG([],[],[-3 -5./2
    ↪ 0],[],(z./2./lmfif).^2)-2.*meijerG([],[],[-3 -5./2 1],[],(z./2./lmfif).^2));
FThV={FThAuss,FThPaeva};
FThf1=FThV{selvec(FThX)};

%$#@!
```

```
% FacHavY

plotFTh='fplot(@(z) FThf1(gradTf(z),vTnf(Tif(z))),[z1 z2])';

%% Coulomb
% FC
% Monopole: Q.*E, no nonsense
FCmon =@(Qpf,nesf,nisf,Ef,lDf) Qpf.*Ef;
% + Dipole term
FCdip =@(Qpf,nesf,nisf,Ef,lDf) Qpf.*Ef+rp.^3.*el.*(nisf-nesf).*Ef.*(1+(rp./lDf).^2./3.
    ↪ /(1+rp./lDf));
FCV={FCmon,FCdip};
FCf=FCV{selvec(FCX)};

plotFC='fplot(@(z) FCf(Qpf(Vpf(Tef(z),Tif(z),nesf(Phif(Tef(z),z),Tef(z)),nisf(Phif(Tef(z)
    ↪ ,z),Tef(z),z,lmfif(Tif(z),siginf(Tif(z))))),nesf(Phif(Tef(z),z),Tef(z)),nisf(Phif
    ↪ (Tef(z),z),Tef(z),z,lmfif(Tif(z),siginf(Tif(z)))),Ef(Phif(Tef(z),z),z),lDf(lDef(
    ↪ Tef(z),nesf(Phif(Tef(z),z),Tef(z))),lDif(Tif(z),nisf(Phif(Tef(z),z),Tef(z),z,lmfif
    ↪ (Tif(z),siginf(Tif(z)))))),[z1 z2])';


%% Sum of the forces
% FS
FS =@(FNf,FThf1,FCf,Fif) FNf+FThf1+FCf+Fif;

% Plot all
plotFS='fplot(@(z) FS(FNf(up,vTnf(Tif(z)),Tif(z)),FThf1(gradTf(z),vTnf(Tif(z))),FCf(Qpf(
    ↪ Vpf(Tef(z),Tif(z),nesf(Phif(Tef(z),z),Tef(z)),nisf(Phif(Tef(z),z),Tef(z),z,lmfif(
    ↪ Tif(z),siginf(Tif(z))))),nesf(Phif(Tef(z),z),Tef(z)),nisf(Phif(Tef(z),z),Tef(z),z
    ↪ ,lmfif(Tif(z),siginf(Tif(z)))),Ef(Phif(Tef(z),z),z),lDf(lDef(Tef(z),nesf(Phif(Tef(
    ↪ z),z),Tef(z))),lDif(Tif(z),nisf(Phif(Tef(z),z),Tef(z),z,lmfif(Tif(z),siginf(Tif(z)
    ↪ ))))),Fif(nisf(Phif(Tef(z),z),Tef(z),z,lmfif(Tif(z),siginf(Tif(z)))),uif(Phif(Tef
    ↪ (z),z)),lDf(lDef(Tef(z),nesf(Phif(Tef(z),z),Tef(z))),lDif(Tif(z),nisf(Phif(Tef(z),
    ↪ z),Tef(z),z,lmfif(Tif(z),siginf(Tif(z))))),uieff(uif(Phif(Tef(z),z)),Tif(z)),bcf(
    ↪ Vpf(Tef(z),Tif(z),nesf(Phif(Tef(z),z),Tef(z)),nisf(Phif(Tef(z),z),Tef(z),z,lmfif(
    ↪ Tif(z),siginf(Tif(z)))),uif(Phif(Tef(z),z))),bcollf(Vpf(Tef(z),Tif(z),nesf(Phif(
    ↪ Tef(z),z),Tef(z)),nisf(Phif(Tef(z),z),Tef(z),z,lmfif(Tif(z),siginf(Tif(z))))),Tif(
    ↪ z)),b0f(Qpf(Vpf(Tef(z),Tif(z),nesf(Phif(Tef(z),z),Tef(z)),nisf(Phif(Tef(z),z),Tef(
    ↪ z),z,lmfif(Tif(z),siginf(Tif(z))))),uieff(uif(Phif(Tef(z),z)),Tif(z))),Vpf(Tef(z)
    ↪ ,Tif(z),nesf(Phif(Tef(z),z),Tef(z)),nisf(Phif(Tef(z),z),Tef(z),z,lmfif(Tif(z),
    ↪ siginf(Tif(z)))),LOGf(lDf(lDef(Tef(z),nesf(Phif(Tef(z),z),Tef(z))),lDif(Tif(z),
    ↪ nisf(Phif(Tef(z),z),Tef(z),z,lmfif(Tif(z),siginf(Tif(z)))))),b0f(Qpf(Vpf(Tef(z),
    ↪ Tif(z),nesf(Phif(Tef(z),z),Tef(z)),nisf(Phif(Tef(z),z),Tef(z),z,lmfif(Tif(z),
    ↪ siginf(Tif(z))))),uieff(uif(Phif(Tef(z),z)),Tif(z)))),[z1 z2])';

%% Net force
% Ft
Fnet =@(FNf,FThf1,FCf,Fif,Fgf) FNf+FThf1+FCf+Fif+Fgf;

plotFnet='fplot(@(z) Fnet(FNf(up,vTnf(Tif(z)),Tif(z)),FThf1(gradTf(z),vTnf(Tif(z))),FCf(
    ↪ Qpf(Vpf(Tef(z),Tif(z),nesf(Phif(Tef(z),z),Tef(z)),nisf(Phif(Tef(z),z),Tef(z),z,
    ↪ lmfif(Tif(z),siginf(Tif(z))))),nesf(Phif(Tef(z),z),Tef(z)),nisf(Phif(Tef(z),z),
    ↪ Tef(z),z,lmfif(Tif(z),siginf(Tif(z)))),Ef(Phif(Tef(z),z),z),lDf(lDef(Tef(z),nesf(
    ↪ Phif(Tef(z),z),Tef(z))),lDif(Tif(z),nisf(Phif(Tef(z),z),Tef(z),z,lmfif(Tif(z),
```

```
    ↪ siginf(Tif(z))))))),Fif(nisf(Phif(Tef(z),z),Tef(z),z,lmfif(Tif(z),siginf(Tif(z))))
    ↪ ,uif(Phif(Tef(z),z)),lDf(lDef(Tef(z),nesf(Phif(Tef(z),z),Tef(z))),lDif(Tif(z),nisf
    ↪ (Phif(Tef(z),z),Tef(z),z,lmfif(Tif(z),siginf(Tif(z))))),uieff(uif(Phif(Tef(z),z))
    ↪ ,Tif(z)),bcf(Vpf(Tef(z),Tif(z),nesf(Phif(Tef(z),z),Tef(z)),nisf(Phif(Tef(z),z),Tef
    ↪ (z),z,lmfif(Tif(z),siginf(Tif(z)))),uif(Phif(Tef(z),z))),bcollf(Vpf(Tef(z),Tif(z)
    ↪ ,nesf(Phif(Tef(z),z),Tef(z)),nisf(Phif(Tef(z),z),Tef(z),z,lmfif(Tif(z),siginf(Tif(
    ↪ z)))),Tif(z)),b0f(Qpf(Vpf(Tef(z),Tif(z),nesf(Phif(Tef(z),z),Tef(z)),nisf(Phif(Tef
    ↪ (z),z),Tef(z),z,lmfif(Tif(z),siginf(Tif(z))))),uieff(uif(Phif(Tef(z),z)),Tif(z)))
    ↪ ,Vpf(Tef(z),Tif(z),nesf(Phif(Tef(z),z),Tef(z)),nisf(Phif(Tef(z),z),Tef(z),z,lmfif(
    ↪ Tif(z),siginf(Tif(z)))),LOGf(lDf(lDef(Tef(z),nesf(Phif(Tef(z),z),Tef(z))),lDif(
    ↪ Tif(z),nisf(Phif(Tef(z),z),Tef(z),z,lmfif(Tif(z),siginf(Tif(z))))),b0f(Qpf(Vpf(
    ↪ Tef(z),Tif(z),nesf(Phif(Tef(z),z),Tef(z)),nisf(Phif(Tef(z),z),Tef(z),z,lmfif(Tif(z
    ↪ ),siginf(Tif(z))))),uieff(uif(Phif(Tef(z),z)),Tif(z))))),Fgf(z)),[z1 z2])';
```

A.6 Redefine parameters

```
% Redefiner redefines final parameters based on new parameter choices.

TiAf=TiAV{selvec(TiAX)};
TeAf=TeAV{selvec(TeAX)};
sigenf=sigenV{selvec(sigenX)};
siginf=siginV{selvec(siginX)};
sf=sV{selvec(sX)};
Tif=TiV{selvec(TiAX)};
Tef=TeV{selvec(TeAX)};
Phif=PhiV{selvec(PhiX)};
uif=uiV{selvec(uiX)};
nisf=nisV{selvec(nisX)};
nesf=nesV{selvec(nesX)};
vTnf=vTnV{selvec(vTnX)};
gradTf=gradTV{selvec(gradTX)};
Ef=EV{selvec(PhiX)};
Fif=FiV{selvec(FiX)};
FNf=FNV{selvec(FNX)};
FThf1=FThV{selvec(FThX)};
FCf=FCV{selvec(FCX)};
```

A.7 Compare parameter options

```
% Plotcompare allows easy comparison of the effect of different options of
% different variables
% Make sure you first ran selectvars.

% First input which variable you want to vary. Type the term as a string.
variable1=input('Which variable? ');
varindex=eval(strcat(variable1,'X'));
varamount=eval(strcat(variable1,'N'));
% Input the function you want to output. Type the term as a string.
variable2=input('Variable to compare: ');
```

```
plotvar=eval(strcat('plot',variable2));
varvec=eval(strcat(variable2,'V'));

% In order to make sure that you will retain your old choices, the changing
% variable will temporarily be stored here.
tempchoice=selvec(varindex);

hold off

for j=1:varamount
    selvec(varindex)=j;
    forcesworking7
    evalc(plotvar);
    disp(varvec{j})
    if j==1
        hold on
    end
end

% Reload your old choice.
selvec(varindex)=tempchoice;
forcesworking7
```

A.8 Compare all options

```
% Plotcompare2 lets matlab compare all possible options for the variables
% against one another. Wild.

% Define a 15-dimensional (no joke) matrix with all possible outputs of the
% model.
startanew=input('Start anew? No(0),Yes(1): ');
if startanew==1
    valuevec=zeros(TiAN,TeAN,siginN,sigenN,sN,PhiN,uiN,nisN,nesN,vTnN,gradTN,FiN,FNN,FThN
        ↪ ,FCN);
else
    load valuevec.mat
end

forcesworking7
dispmax=[TiAN TeAN siginN sigenN sN PhiN uiN nisN];
disp(dispmax);
for i1=1:TiAN; selvec(TiAX)=i1; %#ok<SAGROW>
for i2=1:TeAN; selvec(TeAX)=i2;
for i3=1:siginN; selvec(siginX)=i3;
for i4=1:sigenN; selvec(sigenX)=i4;
for i5=1:sN; selvec(sX)=i5;
for i6=1:PhiN; selvec(PhiX)=i6;
    c=clock;
disp(c(4:6)); % To allow you to see how long the program has been running
for i7=1:uiN; selvec(uiX)=i7;
for i8=1:nisN; selvec(nisX)=i8;
```

```
dispvec=[i1 i2 i3 i4 i5 i6 i7 i8];
disp(dispvec) % To allow you to track the progress of the system.
for i9=1:nesN; selvec(nesX)=i9;
for i10=1:vTnN; selvec(vTnX)=i10;
for i11=1:gradTN; selvec(gradTX)=i11;
for i12=1:FiN; selvec(FiX)=i12;
for i13=1:FNN; selvec(FNX)=i13;
for i14=1:FThN; selvec(FThX)=i14;
for i15=1:FCN; selvec(FCX)=i15;
    redefiner
    % The line below is the 'indicated line'
    Fnetz=Fnet(FNf(up,vTnf(Tif(zar)),Tif(zar)),FThf1(gradTf(zar),vTnf(Tif(zar))),FCf(Qpf(
        ↪ Vpf(Tef(zar),Tif(zar),nesf(Phif(Tef(zar),zar),Tef(zar)),nisf(Phif(Tef(zar),zar)
        ↪ ,Tef(zar),zar,lmfif(Tif(zar),siginf(Tif(zar))))),nesf(Phif(Tef(zar),zar),Tef(
        ↪ zar)),nisf(Phif(Tef(zar),zar),Tef(zar),zar,lmfif(Tif(zar),siginf(Tif(zar)))),Ef
        ↪ (Phif(Tef(zar),zar),zar),lDf(lDef(Tef(zar),nesf(Phif(Tef(zar),zar),Tef(zar))),
        ↪ lDif(Tif(zar),nisf(Phif(Tef(zar),zar),Tef(zar),zar,lmfif(Tif(zar),siginf(Tif(
        ↪ zar)))))),Fif(nisf(Phif(Tef(zar),zar),Tef(zar),zar,lmfif(Tif(zar),siginf(Tif(
        ↪ zar)))),uif(Phif(Tef(zar),zar)),lDf(lDef(Tef(zar),nesf(Phif(Tef(zar),zar),Tef(
        ↪ zar))),lDif(Tif(zar),nisf(Phif(Tef(zar),zar),Tef(zar),zar,lmfif(Tif(zar),siginf
        ↪ (Tif(zar)))))),uieff(uif(Phif(Tef(zar),zar)),Tif(zar)),bcf(Vpf(Tef(zar),Tif(zar
        ↪ ),nesf(Phif(Tef(zar),zar),Tef(zar)),nisf(Phif(Tef(zar),zar),Tef(zar),zar,lmfif(
        ↪ Tif(zar),siginf(Tif(zar)))),uif(Phif(Tef(zar),zar))),bcollf(Vpf(Tef(zar),Tif(
        ↪ zar),nesf(Phif(Tef(zar),zar),Tef(zar)),nisf(Phif(Tef(zar),zar),Tef(zar),zar,
        ↪ lmfif(Tif(zar),siginf(Tif(zar)))),Tif(zar)),b0f(Qpf(Vpf(Tef(zar),Tif(zar),nesf
        ↪ (Phif(Tef(zar),zar),Tef(zar)),nisf(Phif(Tef(zar),zar),Tef(zar),zar,lmfif(Tif(
        ↪ zar),siginf(Tif(zar))))),uieff(uif(Phif(Tef(zar),zar)),Tif(zar))),Vpf(Tef(zar)
        ↪ ,Tif(zar),nesf(Phif(Tef(zar),zar),Tef(zar)),nisf(Phif(Tef(zar),zar),Tef(zar),
        ↪ zar,lmfif(Tif(zar),siginf(Tif(zar)))),LOGf(lDf(lDef(Tef(zar),nesf(Phif(Tef(zar
        ↪ ),zar),Tef(zar))),lDif(Tif(zar),nisf(Phif(Tef(zar),zar),Tef(zar),zar,lmfif(Tif(
        ↪ zar),siginf(Tif(zar))))),b0f(Qpf(Vpf(Tef(zar),Tif(zar),nesf(Phif(Tef(zar),zar)
        ↪ ,Tef(zar)),nisf(Phif(Tef(zar),zar),Tef(zar),zar,lmfif(Tif(zar),siginf(Tif(zar))
        ↪ )))),uieff(uif(Phif(Tef(zar),zar)),Tif(zar)))),Fgf(zar));
    % Or the line above, depending on your perspective.
    % Then change the terms in the two lines below
    Fnetdev=sum(Fnetz.^2)*zstep;
    % Maybe use Fnetz times its complex conjugate to remove complex part.
    valuevec(i1,i2,i3,i4,i5,i6,i7,i8,i9,i10,i11,i12,i13,i14,i15)=Fnetdev;
end; end; end; end; end; end; end; end; end; end; end; end; end; end; end

valsave=input('Save? No(0),Yes(1): ');
if valsave == 1
    save('valuevec.mat','valuevec');
end
```

A.9 Find the best option

```
% Compareanalyser analyses the valuevec that plotcompare2 yields.

disp('What type of mininimum do you want?')
mintype=input('Modulus (1), real part (2), complex (3), real only(4): ');
```

```
% The answers are of the form a+b*i. 1 determines the smallest value of
% sqrt(a^2+b^2), 2 the smallest value of a and 3 the actual value
% calculated by matlab
if mintype==1
    finn=abs(valuevec);
end
if mintype==2
    finn=real(valuevec);
end
if mintype==3
    finn=valuevec;
end
if mintype==4
    finn=valuevec; % Copy valuevec
    finn(imag(finn)~=0)=0; % Set the ones with a complex part to zero
end

finn1=finn(:,:,:,:,5,:,:,:,:,:,:,:,:,:,:); % Make sure only numerically
% calculated sheath can be used.
minval=min(finn1(finn1>0));
minpos=find(finn==minval);
minsize=length(minpos);
minarr=zeros(minsize,15);

for minnow=1:minsize % Try all options corresponding to this minimum.
    foundbool=0;
for id1=1:TiAN
for id2=1:TeAN
for id3=1:siginN
for id4=1:sigenN
for id5=5:sN
for id6=1:PhiN
for id7=1:uiN
for id8=1:nisN
for id9=1:nesN
for id10=1:vTnN
for id11=1:gradTN
for id12=1:FiN
for id13=1:FNN
for id14=1:FThN
for id15=1:FCN
    if foundbool==0 % To avoid calculating this for every combination if indexval is
        ↪ found already
        indexval=1+(id1-1)+TiAN*(id2-1)+TiAN*TeAN*(id3-1)+TiAN*TeAN*siginN*(id4-1)+TiAN*
            ↪ TeAN*siginN*sigenN*(id5-1)+TiAN*TeAN*siginN*sigenN*sN*(id6-1)+TiAN*TeAN*
            ↪ siginN*sigenN*sN*PhiN*(id7-1)+TiAN*TeAN*siginN*sigenN*sN*PhiN*uiN*(id8-1)+
            ↪ TiAN*TeAN*siginN*sigenN*sN*PhiN*uiN*nisN*(id9-1)+TiAN*TeAN*siginN*sigenN*sN
            ↪ *PhiN*uiN*nisN*nesN*(id10-1)+TiAN*TeAN*siginN*sigenN*sN*PhiN*uiN*nisN*nesN*
            ↪ vTnN*(id11-1)+TiAN*TeAN*siginN*sigenN*sN*PhiN*uiN*nisN*nesN*vTnN*gradTN*(
            ↪ id12-1)+TiAN*TeAN*siginN*sigenN*sN*PhiN*uiN*nisN*nesN*vTnN*gradTN*FiN*(id13
            ↪ -1)+TiAN*TeAN*siginN*sigenN*sN*PhiN*uiN*nisN*nesN*vTnN*gradTN*FiN*FNN*(id14
            ↪ -1)+TiAN*TeAN*siginN*sigenN*sN*PhiN*uiN*nisN*nesN*vTnN*gradTN*FiN*FNN*FThN
            ↪ *(id15-1);
        if indexval==minpos(minnow)
```

```
            posvec=[id1 id2 id3 id4 id5 id6 id7 id8 id9 id10 id11 id12 id13 id14 id15];
            for arrindex=1:length(posvec)
                minarr(minnow,arrindex)=posvec(arrindex);
            end
            foundbool=1;
        end
    end
end; end; end; end; end; end; end; end; end; end; end; end; end; end; end
end

% Display the quality and coordinates of the minimum
disp(minval)
disp(minarr)
% Set selvec to the appropriate values
option=input('Which would you like to plot? ');
selvec(TiAX)=minarr(option,1);
selvec(TeAX)=minarr(option,2);
selvec(siginX)=minarr(option,3);
selvec(sigenX)=minarr(option,4);
selvec(sX)=minarr(option,5);
selvec(PhiX)=minarr(option,6);
selvec(uiX)=minarr(option,7);
selvec(nisX)=minarr(option,8);
selvec(nesX)=minarr(option,9);
selvec(vTnX)=minarr(option,10);
selvec(gradTX)=minarr(option,11);
selvec(FiX)=minarr(option,12);
selvec(FNX)=minarr(option,13);
selvec(FThX)=minarr(option,14);
selvec(FCX)=minarr(option,15);

forcesworking7

% Plot the 6 forces
hold off
fFnet=eval(plotFnet); % Blue
fFnet.LineWidth=1.5;
hold on
fFg=eval(plotFg); % Orange
fFg.LineWidth=1.5;
fFN=eval(plotFN); % Yellow
fFN.LineWidth=1.5;
fFi=eval(plotFi); % Purple
fFi.LineWidth=1.5;
fFC=eval(plotFC); % Green
fFC.LineWidth=1.5;
fFTh=eval(plotFTh); % Dark Red
fFTh.LineWidth=1.5;
fFTh.Color=[0.635 0.078 0.184]; % Dark Red
hold off
```

References

[1] Wolfgang Baumjohann and Rudolf A Treumann. *Basic space plasma physics*. World Scientific Publishing Company, 2012.

[2] Diego pmc. Lightning - wikipedia. `https://commons.wikimedia.org/wiki/File:Lightning_over_Oradea_Romania_3.jpg`, 01-12-2008.

[3] CCFE. Plasma confinement - iter. `https://www.iter.org/sci/plasmaconfinement`.

[4] Mike Wehner. We finally know why grapes go nuclear in the microwave. `https://bgr.com/2019/02/19/microwave-grapes-study-plasma-science/`, 19-02-2019.

[5] Niagara Health. How to be safe in the sun. `https://www.niagarahealth.on.ca/site/news/2017/06/17/how-to-be-safe-in-the-sun`, 17-06-2017.

[6] J Winter and G Gebauer. Dust in magnetic confinement fusion devices and its impact on plasma operation. *Journal of nuclear materials*, 266:228–233, 1999.

[7] P Roca i Cabarrocas, A Fontcuberta i Morral, and Y Poissant. Growth and optoelectronic properties of polymorphous silicon thin films. *Thin Solid Films*, 403:39–46, 2002.

[8] John R Taylor. *Classical mechanics*. University Science Books, 2005.

[9] Michael Paul Hobson, George P Efstathiou, and Anthony N Lasenby. *General relativity: an introduction for physicists*. Cambridge University Press, 2006.

[10] James R Welty, Charles E Wicks, Gregory Rorrer, and Robert E Wilson. *Fundamentals of momentum, heat, and mass transfer*. John Wiley & Sons, 2009.

[11] David J Griffiths. Introduction to electrodynamics, 2005.

[12] Michael A Lieberman, Allan J Lichtenberg, et al. *Principles of plasma discharges and materials processing*, volume 2. Wiley Online Library, 2005.

[13] Andr Bouchoule. *Dusty plasmas : physics, chemistry, and technological impacts in plasma processing*. John Wiley & Sons Ltd, 1999.

[14] Av V Phelps. The application of scattering cross sections to ion flux models in discharge sheaths. *Journal of applied physics*, 76(2):747–753, 1994.

[15] Gabriela Veselinova Paeva. Sheath phenomena in dusty plasmas. Technical report, Technische Universiteit Eindhoven Netherlands, 2005.

[16] KP Subramanian and Vijay Kumar. Total electron scattering cross sections for argon, krypton and xenon at low electron energies. *Journal of Physics B: Atomic and Molecular Physics*, 20(20):5505, 1987.

[17] David Bohm. The characteristics of electrical discharges in magnetic fields. *Qualitative Description of the Arc Plasma in a Magnetic Field*, 1949.

[18] K-U Riemann. The bohm criterion and sheath formation. *Journal of Physics D: Applied Physics*, 24(4):493, 1991.

[19] Wolfram Research Inc. Mathematica, Version 12.0, 2019. Champaign, IL.

[20] DUB Aussems, SA Khrapak, İ Doğan, MCM van de Sanden, and TW Morgan. An analytical force balance model for dust particles with size up to several debye lengths. *Physics of Plasmas*, 24(11):113702, 2017.

[21] MD Kilgore, JE Daugherty, RK Porteous, and DB Graves. Ion drag on an isolated particulate in a low-pressure discharge. *Journal of applied physics*, 73(11):7195–7202, 1993.

[22] SA Khrapak, AV Ivlev, SK Zhdanov, and GE Morfill. Hybrid approach to the ion drag force. *Physics of plasmas*, 12(4):042308, 2005.

[23] Vladimir E Fortov, Aleksei G Khrapak, Sergei A Khrapak, Vladimir I Molotkov, and Oleg F Petrov. Dusty plasmas. *Physics-Uspekhi*, 47(5):447, 2004.

[24] LRKRWDR Talbot, RK Cheng, RW Schefer, and DR Willis. Thermophoresis of particles in a heated boundary layer. *Journal of fluid mechanics*, 101(4):737–758, 1980.

[25] Katrin Swimm, Gudrun Reichenauer, Stephan Vidi, and Hans-Peter Ebert. Impact of thermal coupling effects on the effective thermal conductivity of aerogels. *Journal of Sol-Gel Science and Technology*, 84(3):466–474, 2017.

[26] O Havnes, T Nitter, V Tsytovich, GE Morfill, and T Hartquist. On the thermophoretic force close to walls in dusty plasma experiments. *Plasma Sources Science and Technology*, 3(3):448, 1994.